SYNTACTIC ANALYSIS IN A SPEECH UNDERSTANDING SYSTEM

Madeleine Bates

Bolt Beranek and Newman, Incorporated

Prepared for:

Office of Naval Research

August 1975

BOLT BERANEK AND NEWMAN INC
097168
CONSULTING • DEVELOPMENT • RESEARCH

BBN Report No. 3116

SYNTACTIC ANALYSIS IN A SPEECH UNDERSTANDING SYSTEM

Madeleine Bates

August 1975

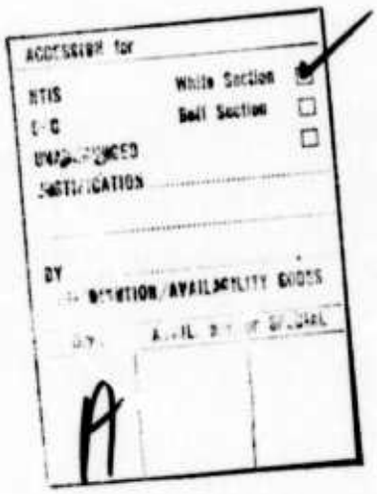
Sponsored by
Advanced Research Projects Agency
ARPA Order No. 2904

This research was principally supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-75-C-0533. Partial support of the author by NSF grant GS-39834 to Harvard University is gratefully acknowledged.

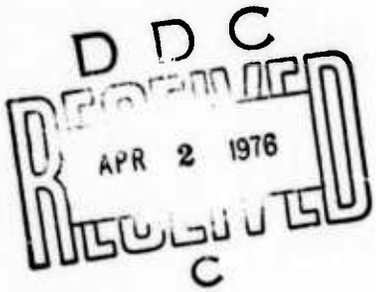
The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

SYNTACTIC ANALYSIS IN A SPEECH UNDERSTANDING SYSTEM

Madeleine Bates



August 1975



Sponsored by
Advanced Research Projects Agency
ARPA Order No. 2904

This research was principally supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract No. N00014-75-C-0533. Partial support of the author by NSF grant GS-39834 to Harvard University is gratefully acknowledged.

SYNOPSIS

This report presents a system for syntactic analysis in the context of a computer system for the understanding of spontaneously spoken English.

When people speak, they make certain assumptions about the ability of their listener to understand them. Human listeners are very good at understanding even very noisy, incomplete, and ambiguous speech when it is presented in a context which allows the listener to draw on his knowledge of the topic under discussion, general knowledge of the world, knowledge of the speaker and of the previous dialogue, and other non-acoustic information. Presumably because of this assumed capability, speakers do not produce an acoustic signal which carries enough information to be decoded into unique phonemes or words on the basis of acoustic information alone.

This implies that no matter how good the acoustic processor of a speech understanding system is, it will not be able to uniquely identify all the words of the utterance, and some other processors will be required to use non-acoustic knowledge, such as syntax and semantics, to fill in and disambiguate the utterance. This report is

concerned with the role of syntax is such a process.

A syntactic processor for speech must be constructed quite differently from one for text input. Given that the words produced from the acoustic information may not be unique, adjacent, or all correct, we will argue that strict left to right processing is precluded. The syntactic processor must be able to predict words or syntactic classes which may fill the gaps in the utterance. It must also identify syntactic structures which may be formed from the partial information which is available.

The syntactic analysis system presented in this thesis is composed of two parts, a grammar and a parser. The parser uses the grammar to process a partial utterance and to make predictions about missing words.

The grammar for the system is written in a modification of the Augmented Transition Network (ATN) formalism developed by Woods [91-93]. The grammar itself began as a modified subset of the previously existing ATN grammar for the LUNAR text question-answering system [97] but has been expanded to include constructions not accepted by the LUNAR grammar such as dates and compound number expressions. The expressive power of the modified ATN formalism is well beyond the finite state, context

free, or augmented context free grammars used by other current speech understanding systems (see Chapter Two for a review of such systems).

The parser for the system is completely different in design from Woods' parser and is the primary contribution of this work. It uses a judicious mixture of top down, bottom up, depth first, and breadth first parsing strategies to take advantage of local information which is available in the input without being drawn into long, erroneous paths which must later be abandoned. It also saves all the information gained while following alternative parsing paths so that any two (or more) parse paths which have a common portion can share the common part without reparsing it. This is true even if the parse paths split before and/or after the common section, and even if the common section analyzes only part of a complete syntactic constituent.

One of the most severe problems faced by a speech parser is the combinatorial explosion of partial parse paths resulting from input in which not all the words in the sentence are available. This system controls the explosion by several methods: the use of a well-formed-substring table to store constituents which have been parsed so that they never need to be re-processed, merging of partial parse paths, attempting

to extend only the best (i.e. most likely) parse paths, building a data base of parse paths which can be shared from one invocation of the parser to the next, and using other sources of information such as semantics wherever possible to help identify most likely paths.

Chapter One describes the nature of speech and the speech understanding process, concluding with a description of the input, output, and processing characteristics which a speech parser must have. Chapter Two reviews several methods for parsing formal languages and English text, and surveys current work in speech understanding with particular emphasis on syntactic capability.

In Chapter Three we detail the ATN grammar formalism and the modifications which have been made to it, and describe the index into the grammar which provides information needed for some right to left processing and for making predictions. A small sample grammar is also given.

Chapter Four gives an overview of the operation of the speech parser, with illustrations, and Chapter Five presents more details of the process, particularly the scoring mechanisms. Annotated examples of the system in operation are given in Chapter Six. Chapter Seven

evaluates the work and suggests extensions and areas of interest for further research.

The appendices present in detail two grammars used by the system and give the vocabulary currently in use, with a breakdown by syntactic categories and features.

TABLE OF CONTENTS

PAGE

APPENDICES

FIGURES

Chapter 1

Introduction

1.1 OVERVIEW

Understanding speech is an extremely complex process which requires the use of many types of knowledge, one of which is syntax. This thesis presents a system called SPARSER which is designed to provide and use the syntactic knowledge necessary to support an artificial speech understanding system.

The remainder of this chapter discusses the nature of the problem and presents the assumptions about other components of the system which will necessarily interact with syntax. Chapter Two discusses various parsing strategies used for formal languages and text understanding systems, and surveys previous and current work in the area of speech understanding. Chapter Three describes the grammar, and Chapter Four presents SPARSER, a speech parser. Chapter Five details the operation of the system, and Chapter Six gives sample results and statistics. The final chapter discusses the strengths and

weaknesses of SPARSER and indicates directions for further research.

1.2 THE NATURE OF SPEECH

There are many types of speech. Words may be spoken in isolation, with long pauses between them. Sentences may be read, usually with strong inflections. Natural, spontaneous speech where the words are spoken together without many pauses may fall anywhere on a continuum from very formal, slow, grammatical speech to very informal. rapid, ungrammatical, conversational speech. We will assume for the remainder of this thesis that unless explicitly stated otherwise "speech" means grammatical speech spoken at a moderate rate with natural inflections and pauses, spontaneously produced but similar to the type of speech produced by reading text.

It is a well documented fact [17, 34, 35, 57] that there is not enough information in the speech signal to uniquely identify the phonemes or words in a normally spoken utterance. This is not due just to the occurrence of homonyms, but to a large number of other factors. Even in speech produced at a normal rate of speed, word and

sentence boundaries are usually obscured (as in "team eating", "team meeting", "tea meeting"), the pronunciation of phonemes is influenced by the surrounding context, phonemes are inserted, deleted, or altered (e.g. "give me" becomes "gimme"), and there is often very little acoustic difference between sounds which are quite different in the ideal phonetic representation.

Besides the ambiguity and error inherent in the acoustic signal, we may safely assume that the acoustic processing component of any artificial speech understanding system will introduce additional errors and ambiguity because of the uncertainty in the process of segmenting continuous speech into units (phonemes, transemes [20], APEL's [89], syllables, words, etc.) and in the identification of those units.

Much of the current knowledge of acoustic-phonetics (which relates acoustic properties of the speech signal to the phonemes which underlie them) consists of rules which are generaive in nature, not analytic. In addition, it is usually not possible to uniquely identify a portion of acoustically processed data as a single phoneme; instead, classification into sets of possible phonemes is all that can be done (see, for example, Schwartz and Makhoul [75]).

For example, it may be possible to determine that a given segment is a nasal without being able to say whether it is an /m/ or an /n/. If such a segment were followed by a segment which could only be classified as a front vowel, and that were in turn followed by an unvoiced stop (either /t/, /k/, or /p/), a schwa, and an /l/, the the word represented by that segmentation and labeling could be either "metal" or "nickel". Determining the segment boundaries themselves is not easy, since a sound like /l/ may appear to be the sequence /ə/ /l/ and vice versa. The recognition of small function words such as "the", "a", "of", "have", "did", etc. is particularly difficult because there is frequently no more than the very slightest acoustic cue to their presence, and such a cue may not be sufficitent to determine the identity of the word.

Lexical retrieval and word matching, the process of scanning an acoustically processed utterance to determine (with the help of a dictionary) what words seem to be there is a non-trivial task which has been discussed in detail elsewhere [42, 71]. Basically, the process of comparing the ideal phonemic representation of a word against a portion of the output of an acoustic processor results in (at least) a score which indicates how well the

expected pronunciation matches the acoustics.

Because of the errors produced by segmentation and labeling, we cannot assume that the correct words will always score best or even high. Special matching rules which attempt to compensate for particularly poor acoustic information at the beginning or end of an utterance may result in matching such phonetically diverse words as "did", "give", and "been" in the same portion of the utterance while missing the correct word "have".

The lexical retrieval process could return the result of matching every word in the system's vocabulary at every point in the utterance but this time-consuming process would produce a vast amount of information, most of it useless. Clearly a better approach would be to limit the number of words returned by discarding all those which score below some given threshold. If the threshold is set too low, a great many spurious words will be found, and there is no guarantee (unless the threshold is zero) that all the correct words will be found in their proper places. The higher the threshold is set, the fewer words (both correct and incorrect) will be found. The optimal threshold will achieve a balance between accuracy and precision. It should not be necessary to retrieve all the

correct words, as long as those which are found are sufficient to suggest those which are missing.

Preliminary results of the BBN speech understanding system (reported by Woods in [94]) indicated that the ratio of correct to incorrect words found can be expected to be range between 1:20 and 1:50, with about 55-60% of the actual words found. More recent results using a different word matching scheme on acoustic data which was segmented and labeled by hand rather than by machine [41] show that this ratio can be lowered to about 1:3 with 63% of the words found if that level of acoustic segmentation can be reached automatically.

We conclude that acoustic and lexical processing of a spoken utterance will result in the discovery of a number of likely word candidates at many places in the utterance. Most of these words will be spurious. This is illustrated in Figure 1.1 by a structure called a word lattice which shows schematically that many words may initially appear to be present in a simple utterance. In this representation the numbers along the horizontal scale are segment boundary points in the utterance which roughly correspond to points in time (but not exactly, since two or more boundary points may be used for the same instant

in time in order to prohibit phonologically incompatible segments from appearing to be juxtaposed).

This word lattice was produced by the lexical retrieval component of the BBN speech understanding system from an utterance which had been segmented and labeled by hand under conditions designed to simulate the performance of an automatic segmenter and labeler.

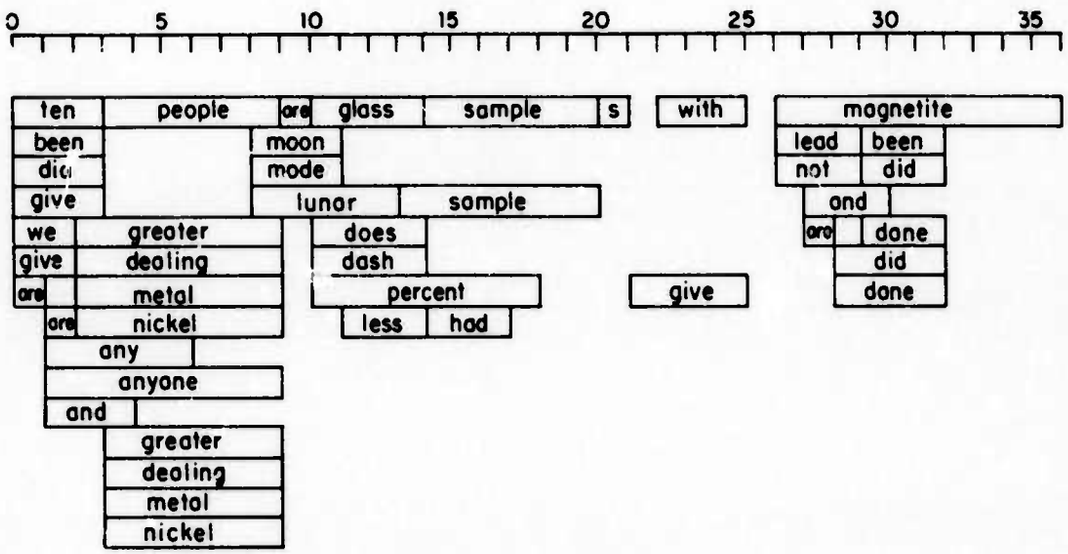


Figure 1.1

A Word Lattice

Sentence: Give me all glass samples with magnetite.

1.3 THE NATURE OF THE SPEECH UNDERSTANDING PROCESS

We do not attempt here to say that the way a particular artificial system works implies something about the way that people understand language. Nor is there enough known about the way people understand language to say that SPARSER is modeled after the human process. However, human beings have been engaged for millions of years in the process of oral communication and presumably have gradually optimized themselves for the task (or perhaps have optimized the task to suit themselves). It is reasonable to suppose that any effective speech understanding system would have some features in common with the way people process speech. We shall therefore justify some of our assumptions concerning the nature of the task and the relation of the syntactic component to the rest of the system by reference to human behavior, without, however, claiming any further similarity.

When people speak naturally and informally they frequently make grammatical mistakes, yet they are easily understood. This would indicate that a syntax-driven system for speech understanding (which would accept only input meeting rigid syntactic requirements) might be adequate for a limited application, but would not be

extendible to more natural, conversational speech. Since a number of word candidates are likely to be found throughout the utterance, it may be fruitful to be able to select a subset of them on semantic, pragmatic, or prosodic grounds as well as syntactic, depending on which cues seem most robust. A system in which syntax was one of a number of equally important components contributing to the understanding of a sentence would be more flexible than one which was totally or primarily syntax driven.

People use extensive knowledge in order to comprehend spoken utterances and it is now generally accepted that any successful speech understanding system must also use a number of knowledge sources in combination [22, 57]. It is not enough to follow the paradigm of "segment the acoustic signal into phonemes, then identify the phonemes and words, then parse the sentence, then interpret the structure." Acoustics alone cannot be depended upon to provide a unique (or even only slightly ambiguous) segmentation of the input stream into words. Also, all the words in the utterance will not necessarily be discovered by acoustic information alone.

Why, in particular, is syntax needed to understand speech? Clearly, it is the grammatical structure of "John hit Bob" which differentiates its meaning from that of "Bob hit John" or "John was hit by Bob". Similarly, consider the syntactic case difference between "he" and "him" in "The man who knew him was going left" and "The man who knew he was going left," [3] where the case of the pronoun immediately signals whether it is simply the object of "knew" or the subject of an embedded complement. Working from the premise that one is more likely to make a grammatical statement than an ungrammatical one, in a situation of lexical ambiguity one may use syntactic consistency to decide between "the cat sin the tree" and "the cat's in the tree". And finally, the existence of two or more syntactic structures indicates ambiguous sentences or constituents such as "He gave her cat food" and "the boy with the cat in the tree" (who is in the tree?). It may be necessary for some syntactic structure to be built for these ambiguous strings in order for some other knowledge source to choose between them. If the syntactic ambiguity cannot be resolved, it means that the utterance is truly ambiguous, in which case the system should perhaps ask for clarification in the same way that a human listener would do.

Syntactic consistency should not be the only factor in determining the utterance however, since it is possible (even likely) to be able to find a syntactically consistent interpretation of the utterance which is wrong. Because there will be many possible words for many portions of the utterance, there would be too many syntactically correct but meaningless combinations of words to justify requiring a complete parse of the input before any semantic processing is done. In order to obtain a word lattice complete enough to parse (i.e. with at least all the content words present) current experience indicates that about 80 words would have to be considered for an average 7 word sentence [41]. Even in the much smaller word lattice of Figure 1.1 it can be seen that there are numerous short sequences which are syntactically but not semantically valid (e.g. "Ten people are glass samples with magnetite", "glass samples give magnetite", "lunar samples give magnetite", "samples give lead", "people are percent", etc.).

1.4 THE NATURE OF THE SYNTACTIC COMPONENT

As a consequence of the nature of speech and speech understanding, the syntactic component of a speech understanding system must deal with input of a different nature than text parsers and must have different operational characteristics as well.

1.4.1 The Input

The input to a parser for speech cannot be a string of uniquely determined words but must be something like a lattice of words (see Figure 1.1). When the parser wants the "next word" of the input it must be able to deal with a list of possible words and must be prepared to cope with the possibility that the correct word is not included in that list. It may also be the case that one or more words that the parser has accepted are wrong. Frequently no usable word can be found at one or more places in the utterance, so the parser must also be able to deal with gaps in its input, for example by predicting one or more words which would be syntactically consistent with the current interpretation of the utterance.

When processing text, a parser can reasonably take advantage of a number of typographic indicators such as punctuation marks (a period to delimit a sentence, commas to disambiguate certain complex conjunction constructions, etc.), capitalization (to indicate the start of a sentence or to distinguish proper nouns such as "Pat" from other words such as the noun or verb "pat"), italics, underlining, quotation marks, and parentheses. (To illustrate the importance of these factors to comprehension, consider the unpunctuated string: "that which is is that which is not is not is not that so" which if correctly punctuated is a grammatically correct sequence of sentences [3].) All of these cues are missing in speech. They are compensated for by the use of pauses, stress, changes in duration, pitch, and loudness, and other prosodic features.

Unfortunately the current lack of knowledge about the acoustic correlates of prosodic features makes it almost impossible to use this rich source of information in speech understanding systems. Current speech parsers must cope with the increased ambiguity resulting from this lack of information, and if designed with foresight, should be easily extendible to use prosodic information when it becomes available. Section 7.2 discusses the issue of

prosodies in more detail.

1.4.2 The Output

In most systems which work with natural language the purpose of the parser is to provide a representation of the syntactic units of the input and their relationsnips to one another. This representation is frequently a "deep structure" tree which may then undergo semantic analysis or interpretation. The creation of a self-contained syntactic structure is not absolutely mandatory if enough semantic and interpretive processing is done together with the parsing, but in any case the syntactic component must be able to confirm that the input is grammatically correct, to detect ambiguities, and to identify syntactic relationships between syntactic groupings of words. We will assume that some structure for it is also produced.

A parser for speech, however, must do more than this. It must aid in selecting a syntactically well-formed sequence of words from the many sequences of words which are possible in the word lattice. It must be able to ask questions of and answer questions from other knowledge sources. For example, upon discovering that a certain sequence of words can be a complete constituent such as a

noun phrase, it may ask for semantic analysis to determine whether the constituent is meaningful. On the other hand, semantics may have already made a supposition about the relationship between two words and the syntactic component may have to determine whether or not that relationship is borne out by the syntactic structure. For example, if the words "analysis" and "iron" are found in the utterance with a small gap between them, a good semantic hypothesis would be that this portion of the sentence is about an analysis of something to see if it contains iron. If after the gap between the words is somehow filled, the syntactic component parses "analysis for iron" the semantic hypothesis is borne out; if it produces "analysis in iron", it is not.

Text parsers are usually designed on the assumption that the words given as input will form a grammatical sentence, so the duty of the parser is merely to determine the structure(s) of the sentence. A speech parser, however, must know that some (in fact, many) of its potential input sequences will be ungrammatical, not because the original utterance was ungrammatical but because some combinations of words which appear to be recognized from the acoustic signal are incorrect. The speech parser should be able to detect and reject those

sequences as early as possible.

Another goal of any speech parser must be to predict words or syntactic categories which could fill gaps in the word lattice. The type and correctness of the predictions which can be made depend on the nature of the grammar being used and the amount of context which is taken into account when making the predictions.

1.4.3 The Processing

Due to the lexical uncertainty inherent in any acoustic analysis and the fact that important words may not be retrievable by acoustics alone, it cannot be assumed that syntactic processing can process strictly left to right (or right to left) through the utterance. Long content words are more reliably identified by acoustics and more easily verified by semantics or pragmatics than short words or function words or words which are garbled at the beginning or end of the utterance. Peculiar phonological behavior occurs at the beginning and the end of an utterance, as a result of the speaker "tooling up" to speak or "tailing off". This makes those portions of the utterance particularly vulnerable to error in lexical recognition. The usually

good advice: "Begin at the beginning and go on till you come to the end: then stop." [13, p.158] does not apply to speech understanding. Thus syntactic processing must begin with whatever reliable anchor points can be found and work "middle out" to fill in the gaps.

The control structure of a speech parser must be a combination of the conventional top down and bottom up approaches: top down in order to make predictions and bottom up in order to minimize errors propagated by dependence on incorrect context. (This issue is discussed in further detail in Chapters Two and Five and in Woods [95].)

If a complete, connected sequence of words could be given to a parser, the number of syntactic alternatives which must be considered is limited by the fact that the surrounding context (particularly the left context, if processing is left to right) limits the number of ways in which an element can be considered. If there is no surrounding context, the possibilities increase. For example, consider the sequence "man eating" which can be part of a number of different constructions:

A man-eating shark.

A man eating an omelet.

A snow man, eating utensils, and several frogs.

In the sky I saw a cloud man, eating lunch on a hill.

If all syntactic possibilities were considered for every small bit of the possible utterance, the resulting combinatorial explosion would preclude obtaining the correct analysis in any reasonable time. Thus the syntactic component must limit the number of syntactic alternatives generated, or at least appropriately factor them or treat them implicitly rather than explicitly, and it must develop the correct alternatives early.

The body of this thesis presents a syntactic system which has the above-mentioned characteristics.

Chapter 2

Review of Parsing Methods and Systems

2.1 INTRODUCTION

This chapter reviews a number of parsing methods which have been developed for formal languages, natural language in text form, and spoken language. The main body of this thesis is concerned only with the analysis of speech, but since much of the terminology and some of the techniques which are used in the other two areas can be carried over to the speech domain, they are of interest here.

2.2 FORMAL LANGUAGES

For the hierarchy of formal languages -- finite state, context free, context sensitive, and recursively enumerable -- there have been developed a large number of parsing algorithms (see for example [1, 33]). We will concentrate here on context free languages (since, interestingly, they are more adaptable to natural language than formal context sensitive languages, partly because of the efficient parsing algorithms which are available for

them). Specifically, we will consider two basic approaches to parsing them: top down and bottom up.

It is usually said that top down means building a deep structure tree by starting at the root node and working down, and that bottom up means starting from the leaves of the tree and building up to the root, but this is somewhat misleading because it does not distinguish between the flow of control of the parser and the order in which the parse tree is constructed. (A recognizer which does not build a structure at all may still be said to be top down or bottom up.)

Virtually all systems actually construct the tree by forming the smallest constituents (near the leaves) first and then combining them in larger and larger groups until the entire tree is built. The basic process which any parser or recognizer goes through is to determine the sequence of rules in the grammar which were applied to generate the string. For our purposes, we will say that if the parser discovers this sequence in the generation order, it is top down, if it discovers it in reverse order, it is bottom up. We will now describe typical top down and bottom up algorithms and discuss their potential advantages and disadvantages with regard to speech processing.

2.2.1 Top Down

Top down parsers are usually left to right (that is, they process the input string from left to right), predictive, and either breadth or depth first. A parser which is depth first attempts at each step in the derivation to choose one rule of the grammar which was used to generate the input string. When an error indicates that the derivation sequence obtained thus far is wrong, the process must "back up" to the last choice point and choose another rule. A breadth first processor applies all possible rules at each step and thus finds all possible derivations in parallel. A predictive algorithm uses the information gained by processing part of the input string to predict what symbol(s) of input will come next. In other words, the next symbol of input is processed only in the context of the previous input, so the context can actually influence the way in which the next symbol is seen.

A strictly top down, depth first, left to right parser begins with the root node S of the grammar and chooses a rule S --> $X_1X_2...X_n$. If $X_1...X_i$ are terminals, they must match the first i characters of input. If X_j is the first nonterminal in $X_1X_2...X_n$, a rule X_j --> $X_{j1}...X_{jk}$ is chosen. The process repeats, expanding the leftmost nonterminal in the current

sentential form and matching the input against initial terminal symbols until either (a) the entire input has been matched and there are no more nonterminals to expand or terminals to match, in which case the parse has been successful or (b) the input does not match the initial terminal characters. In case (b), the process backs up to the previous sentential form and tries to choose a different rule to replace the leftmost non-terminal. If none is available, it backs up another step and tries again. The parse fails if all backup possibilities are exhausted.

This method is somewhat wasteful in that it may require re-parsing a constituent several times if it is "backed over" several times in order to correctly parse input to the left of the constituent. One way to remedy this is by the use of a Well-Formed-Substring Table (WFST) in which all constituents and their boundaries are placed. Then, whenever the parser begins to parse a constituent of a given type, the WFST can be checked to determine if this has already been done, and if so, the constituent can be used directly from the WFST without re-parsing. (An example of this type of analysis as applied to natural language is contained in the Harvard predictive analyzer [43]).

Earley's algorithm [21, 1 p.320] for context free languages is an example of a basically top down, breadth first, left to right, predictive parser which incorporates the idea of a WFST. It constructs, for each position in the input string, a state set of items which represent all the states in which a non-deterministic pushdown automaton could be at that position in the string. That is, it carries in parallel all possible paths of the derivation. In place of a push down stack to keep track of recursion, each state contains a pointer to the state set containing the state (or states) which caused the last push. (This method incidently allows left recursive grammars, which are usually the bane of top down parsers, to be handled gracefully.) When all the entries which can be made in the current state set have been made, one or more states show which terminal and non-terminal symbols can appear in the input at that point. Since the process is breadth first, all possible next symbols are predicted.

In an efficient implementation of Earley's algorithm, the top down process need not be strictly followed since it is possible to compute in advance for each non-terminal symbol the set of allowable initial terminal symbols which can result from expanding that non-terminal. Then it is possible to leap down any number (possibly infinite) of levels of recursion to the input, and once a bottom-most

rule is completed it can be used, bottom up, to select the rules which could have produced that constituent. Earley's algorithm illustrates an important principle: the merging of information in parse paths so that if there are alternative derivations (as for ambiguous strings) which have a number of steps in common, the work required to process the common part need only be done once.

The ability to predict one or more acceptable next symbols of input based on what has already been processed, and then go to the actual input to verify the prediction, is one of the strongest advantages of a top down system. In an environment (such as spoken English) where the "next symbol" is not uniquely determined but is a set of possible symbols, this predictive ability may be used to screen out some or all of the erroneous next symbols or to predict symbols to fill a gap.

Unfortunately, the other side of this coin is a big disadvantage for top down systems. With errorful input, if an error has been made in choosing one of the elements of the context, the prediction depending on that context may screen out the correct symbol from the set of possible next symbols. This may cause the parser to waste considerable time thrashing around in the wrong input, and little if any useful information about the correct string will be gained.

2.2.2 Bottom Up

Bottom up techniques begin with the leaves of a parse tree and find the derivation sequence in reverse, ending with the root node. Such an approach is typified by Cocke's algorithm [1 p.314]. It processes the input by first considering all possible substrings of length one, forming all possible one-word constituents and placing them in a table. Then, using this information, all pairs of adjacent words (and constituents) are considered and all two-word constituents are formed and put in the table. Then all adjacent three-, four-, five-, ... word substrings are considered until the length of the string is reached.

This method is neither left to right nor right to left and has the advantage of working with isolated sections of the input so that an error at one point will not prevent a correct analysis of another portion of the string. Although each constituent need be parsed only once because the tabular method of parsing builds up a WFST, the process unfortunately recuires that all possible parsings of all substrings of the input be found in parallel -- a procedure which is enormously wasteful of space and time even when a single string is being processed.

For speech, the multiple words produced by an acoustic analyzer together with the multiple syntactic categories for many of those words and the multiple ways they can be syntactically combined when only very local context is used, exacerbate the problem to such an extent that a totally bottom up speech parser would be unreasonably slow.

If every word had only one part of speech we could get some idea of how small word groups are likely to combine by looking at small groups of syntactic classes. For example, taking all possible pairs from a reasonable set of 20 parts of speech, are there any pairs which cannot occur in some grammatical sentence? The answer is no, and in fact most pairs can be used in several ways. For many, if not all, triples, the same situation holds. In fact, one can even dig up pathological examples to illustrate unlikely combinations such as having five prepositions in a row: "What did you bring that book I didn't want to be read to out of up for?" [3].

2.3 SYSTEMS FOR ENGLISH TEXT

A large number of systems have been developed which either parse, or parse and interpret, English sentences (especially questions) in printed form. No attempt will be made here to review them all, since a number of excellent reviews are available elsewhere [9, 39, 77, 78, 86]. Instead, a few systems have been chosen for discussion which represent widely different approaches to the problem, and they are described and discussed with particular emphasis on their possible adaptability to speech input.

2.3.1 The Transformational Approach

The most popular model of English in modern linguistics is that of transformational grammar [14-16]. There are many schools of transformational grammar and much dissent on the details of the various approaches, but they basically agree that a transformational grammar consists of at least two parts, a base component and a transformational component. The base component is a set of context free (or sometimes context sensitive) phrase structure rules which operate to produce a deep structure tree. This tree is then processed by the transformational component which is a sequence of structure changing transformations. Each transformation has three parts: a

structural description which is a template to match the tree to be transformed, a set of conditions which must also be satisfied (for example, that two subtrees specified by the template be identical), and a structural change which may insert or delete branches and nodes in the tree and move subtrees from place to place. The transformations are ordered in cyclic sets and may be obligatory, optional, repeatable, or non-repeatable. After some number of transformations, the surface string is formed by reading the leaves of the final transformed (surface structure) tree from left to right.

This approach is primarily a generative one. Attempts have been made to reverse the process, but they have not been particularly successful, in part because of their slowness and inefficiency. Petrick's transformational parser [61-64] uses a specially written context free grammar to parse the input string into one or more possible surface structure trees. Then a series of inverse transformations are applied to undo the effects of the transformational component. Finally any tentative deep structures produced are verified by the transformational phrase structure component. The system is fairly slow, using 38 and 129 seconds to parse sentences which were generated by 14 and 31 forward transformations, respectively.

Another attempt at transformational parsing was developed by Mitre Corporation [54, 99]. It is similar to Petrick's but is even more ad hoc. The surface structure grammar generates all but not only valid surface structures, so some of the original trees obtained may be wrong and cause wasted effort in the inverse transformation phase. The surface structure grammar cannot be derived from the original grammar, nor are the inverse transformations exact inverses of the original transformations -- they were written taking advantage of the grammar writer's knowledge of the overall system.

Neither system would be adaptable to the kind of fuzzy, partial input available in the speech domain because of the number of levels involved (each of which must operate on an entire string or tree) and the combinatorial problems, which are extreme even when the input is known exactly. If all the complete tentative surface structure strings are enumerated and for each one all possible inverse transformations applied at each step, the number of paths being followed can grow exponentially. Any transformation which may have been applied in the generation of the sentence must cause the analysis to split into two paths, one which inverts the transformation and one which does not. The inherent ambiguity of speech input would enormously increase the number of possible

inverses at every step because of the much larger number of possible strings.

2.3.2 String Analysis

String analysis was formulated first by Harris [32], formalized by Joshi [36], and further developed by Sager [73, 31]. The system to be discussed here is Sager's since it embodies the most complete and best documented system using the technique.

Linguistic string theory defines a number of elementary strings in terms of syntactic categories, for example "Noun Tensed-verb" forms an elementary string which can be realized as "Dogs bark." Any sentence string can be made more complex by inserting an adjunct (modifier) string to the left or right of an element of the sentence. (e.g. "Little" can be a left-adjunct to "dogs" and "at mailmen" can be a right-adjunct to "bark.") These rules can be formulated as context free rules. To allow checks for number agreement and other context sensitive effects, each syntactic category may have a number of sub-categories which represent attributes or features such as "plural" and "human". In addition, Sager's system has associated with each rule one or more restriction tests which may look at the sub-categories of the words in the phrase structure tree and test the well

formedness of the tree.

Sager's parsing system consists of three parts: a dictionary, a set of context free rules, and a set of restriction tests. The dictionary has 8000-9000 entries, but contains all inflected (regular as well as irregular) forms, so that the base vocabulary of root forms is actually much smaller. There are 25 major syntactic categories (such as Noun, Verb, Verb with "-ing" Suffix, Past Participle, etc.) and 120 sub-categories.

The 200 grammar rules are used to segment the input string into elementary word strings. Whenever a node is added to the parse tree, the relevant restrictions are interpreted (providing the necessary context sensitive checks) and the path is continued or aborted depending on the success or failure of the restrictions. The parser is top down, left to right, and depth first, but automatically does all backup so that all possible parses of the input are found.

The system has been applied to random sentences taken from literature in the field of pharmacology and is reported to be 60-80% successful, parsing reasonably long sentences in the order of seconds. No general characterization of the scope of the grammar is given and it is difficult to assess whether the success rate given

above results from the somewhat limited style in which most scientific publications are written. It should be possible to adapt this model to speech parsing if a way were worked out to apply the restrictions in an ambiguous, incomplete environment.

2.3.3 Chart Parsing

The MIND system [38] is a set of tools for linguistic processing. The parser for the system is a modification of Cocke's algorithm, which was described in Section 2.2.2 above. It uses a data object called a chart to record possible transitions from one point in the input to another. The chart is a directed graph with vertices and labeled edges. Edges represent mutually exclusive alternatives, each of which represents a transition from one vertex to another and is labeled with an analysis of the portion of the input spanned. See Figure 2.1 for an example of the parsing of the ambiguous string "high ball."

The method of parsing is right to left (!) and bottom up. Like Cocke's algorithm, it finds not only all possible parses of the whole sentence in parallel but also all possible parses of all substrings of the sentence, whether they can be used in a larger constituent or not. On the positive side, the chart acts like a built-in WFST

and provides automatically for sharing of constituents once they are built so no re-parsing is ever necessary.

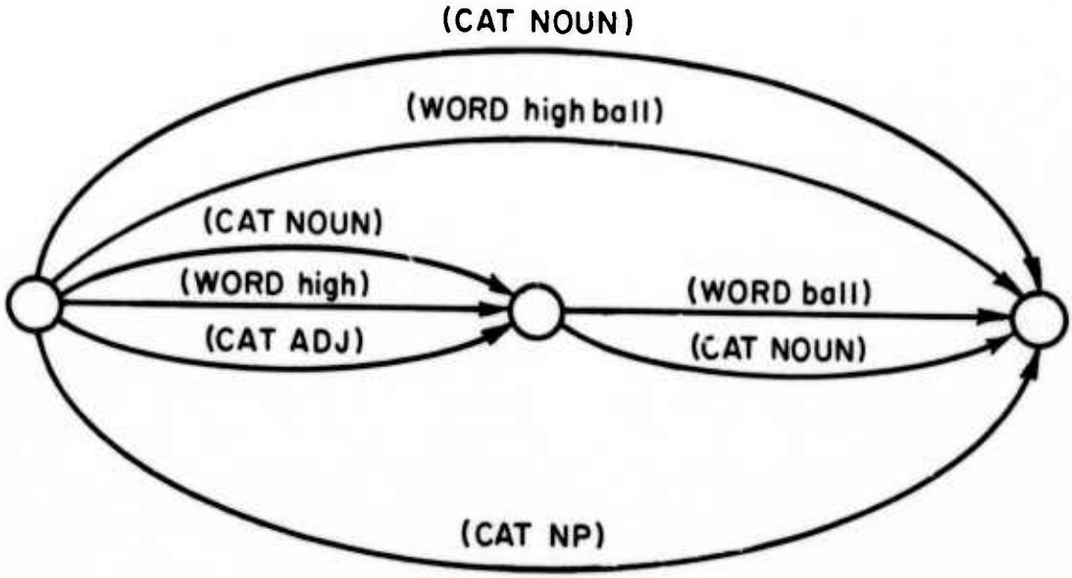


Figure 2.1

Sample Chart for "high ball"

The technique of using a chart was expanded upon by Kaplan [37], resulting in a system for writing parsers called GSP, a general syntactic processor. GSP can still find all parsings in parallel but is not limited to strictly bottom up processing. In fact, Woods' top down parser for transition network grammars (see Section 2.3.4 below) has been implemented in GSP, as has Kay's algorithm.

Kay's strictly bottom up method would not be useful for speech input for the reasons given in Section 2.2.2 above, but Kaplan's system might provide a framework for writing a speech parser. The complexities which would have to be represented in the chart are mind boggling, however, and it is difficult to assess the combinatorial problem involved.

2.3.4 Transition Network Grammars

Augmented transition network grammars (ATN's), the grammar model used as a basis for this thesis, were developed by William Woods [91-93], although similar but less well developed models appear in earlier work by Thorne, Bratley, and Dewar [80] and Bobrow and Fraser [10]. A transition network grammar looks like a finite state transition diagram in that it is a directed graph with labeled states and labeled arcs, a distinguished start state, and a set of distinguished final states. The label on an arc indicates the type (usually syntactic category) of input which will allow a transition to be made to the next state. However, the network permits recursion. That is, the label on some arc may call for a structure (constituent) created by recursively re-applying the network beginning with an indicated initial state.

A basic transition network grammar as described above is equivalent to a context free grammar or a pushdown store automaton. To give it additional power (up to the power of a Turing machine), each arc is augmented with a test and a sequence of actions to produce an Augmented Transition Network (ATN). The test associated with an arc must be satisfied (in addition to the label) for the arc to be taken, and the actions are to be executed as the arc is traversed. These actions construct pieces of tree structure and keep them in registers, which may be thought of as local variables. Register contents are available on subsequent arcs and can be combined, copied, changed, and added to, as more of the tree structure is built.

This very general mechanism provides a transformational capability which can produce deep structures of the same sort as those of a transformational grammar. While the arrangement of states and arcs reflects the surface structures of possible utterances, the actions on the arcs permit elements of input and constituents to be re-arranged so as to produce a deep structure. Figure 2.2 gives a diagram of an ATN for simple passive sentences.

In this diagram, the states are represented as labeled circles (double circles for an initial state) and the arcs as directed arrows between the states. Above

each arc is an indication of the condition which must be true for the arc to be taken (e.g. a word of category Verb or a constituent obtained by applying the grammar beginning at state NP/ to obtain a noun phrase). Below each arc (or in the indicated footnote) is a description of the tests and/or structure building actions on the arc; the symbol * represents the current item of input -- a word or constituent, and structure building functions are replaced by a schematic representation of the structure produced. A more detailed description of ATN grammars will be given in Chapter Three below.

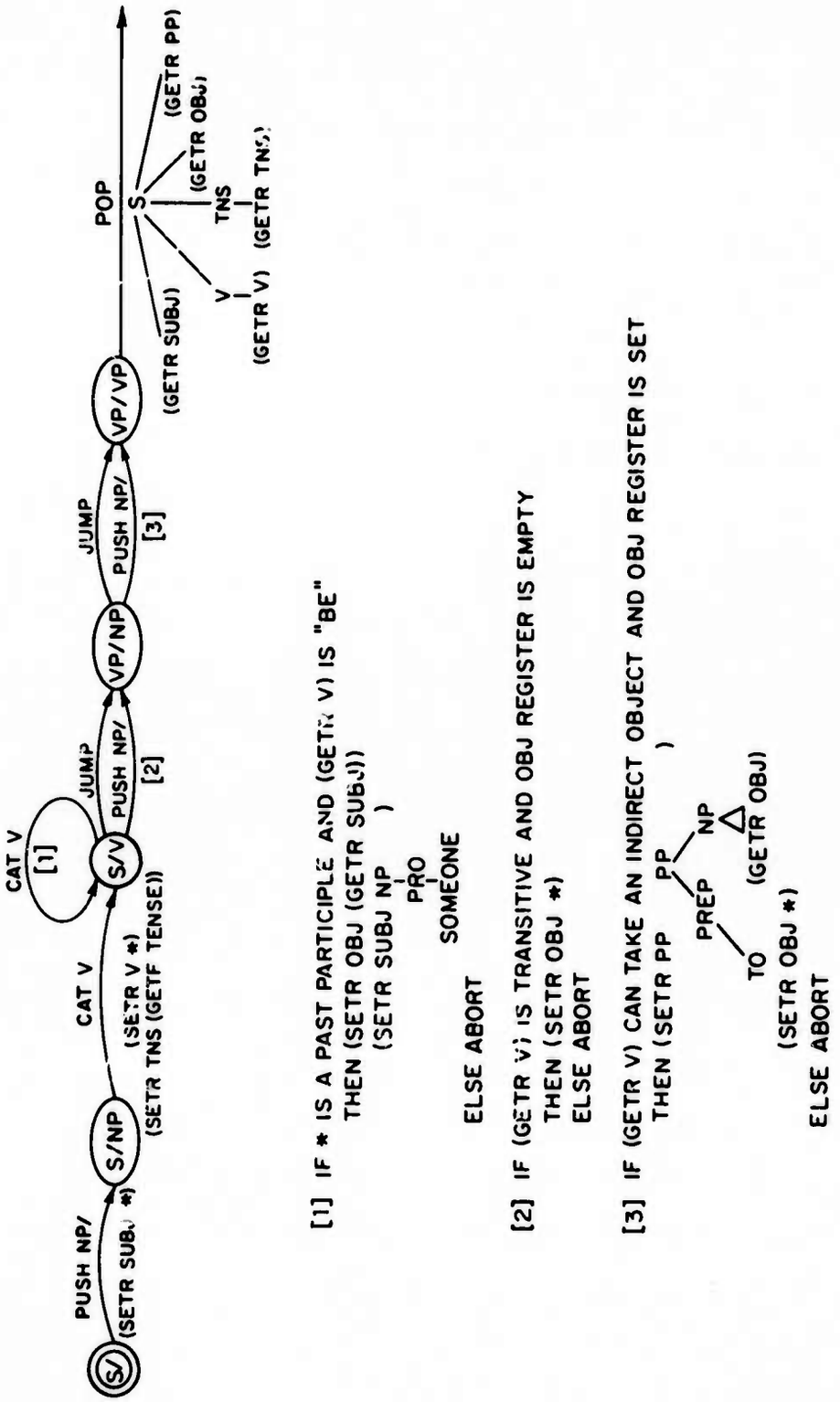

PUSH NP/
(SETR SUBJ *)
S/NP
CAT V
(SETR V *)
(SETR TNS (GETF TENSE))
S/V
CAT V
[1]
JUMP
PUSH NP/
[2]
VP/NP
JUMP
PUSH NP/
[3]
VP/VP
POP
S
(GETR SUBJ)
V
(GETR V)
TNS
(GETR TNS)
(GETR OBJ)
(GETR PP)
[1] IF * IS A PAST PARTICIPLE AND (GETR V) IS "BE"
THEN (SETR OBJ (GETR SUBJ))
(SETR SUBJ NP)
PRO
SOMEONE
ELSE ABORT
[2] IF (GETR V) IS TRANSITIVE AND OBJ REGISTER IS EMPTY
THEN (SETR OBJ *)
ELSE ABORT
[3] IF (GETR V) CAN TAKE AN INDIRECT OBJECT AND OBJ REGISTER IS SET
THEN (SETR PP PP)
PREP
NP
TO
(GETR OBJ)
(SETR OBJ *)
ELSE ABORT

Figure 2.2

It should be noted that the type of structure built is relatively independent of the sequence of states and transitions in the grammar. An ATN could be written to produce case frame descriptions, stratificational analyses, or dependency structures.

The transition network may be viewed as a grammar, as a non-deterministic parser, or as a non-deterministic generator. When viewed as a parser, it is left to right, depth or breadth first, and top down, but bottom up information is easily accessible. When viewed as a grammar, however, it becomes an almost neutral description of a language. Additionally, the distinction between the arc type and the arc actions leads to an easy separation of local information (pertaining to the current word) from context sensitive information.

Another characteristic of ATN grammars, one which strongly suggests its suitability for use in speech understanding, is its efficient merging or factoring of portions of an analysis common to several paths. One can look at a ATN as a model of a context-free grammar in regular expression form (with conditions and actions added to extend the power of the model). Thus a regular expression rule such as X -> (A)BC*D can be represented by an ATN as shown in Figure 2.3.

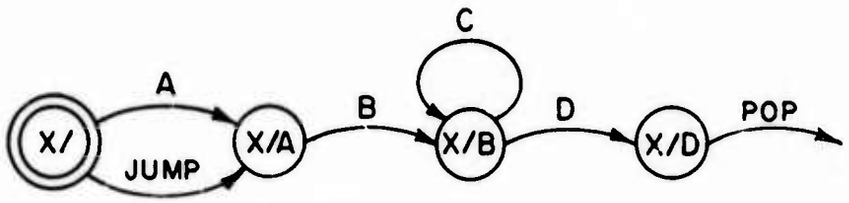


Figure 2.3

Another Simple ATN

Woods says [92 p. 600]:

> The merging of redundant parts of rules not only permits a more compact representation but also eliminates the necessity of redundant processing when doing the parsing. That is, by reducing the size of the grammar representation, one also reduces the number of tests which need to be performed during the parsing. In effect, one is taking advantage of the fact that whether or not a rule is successful in the ordinary context-free grammar model, information is frequently gained in the process of matching it (or attempting to match it) which has implications for the success or failure of later rules. Thus, when two rules have common parts, the matching of the first has already performed some of the tests required for the matching of the second. By merging the common parts, one is able to take advantage of this information to eliminate the redundant processing in the matching of the second rule.

This efficient merging can be achieved for various

parsings which are not identical but are merely similar. That is, information can be stored in registers (and subsequently tested by conditions on other arcs) which would otherwise have to be remembered implicitly by the state of the network. This factoring makes ATN's extremely attractive as a possible grammar form for a speech parser, since some of the combinatorial problems are immediately reduced by taking advantage of the merging capabilities of the grammar.

2.4 SYSTEMS FOR SPEECH

In the past few years there has been a flurry of activity in the field of automatic speech understanding, resulting in a number of different systems. Several of these systems are briefly described here with particular attention to their syntactic capabilities. A more complete survey of these and other systems can be found in Wolf [90].

2.4.1 Vicens-Reddy

The first really workable system to understand continuous speech was developed at Stanford University in the late 1960's [82]. There were several different versions of this system designed for various vocabularies

(carefully selected for content and size) and different numbers of speakers. It achieved 85% correct interpretation (95% correct word recognition) for speakers (for whom the system was specifically tuned) who uttered short sentences composed from a carefully chosen vocabulary of 16 words.

The highly constrained context free grammar would admit only 192 sentences. The controlling program also capitalized upon particular semantic constraints imposed by the limited task domain of block movement, such as "If the sentence starts with 'PICK-UP' then 'BLOCK' must appear somewhere in the sentence", and "Words indicating location can occur only after the word 'BLOCK'".

Although this system was extremely limited in all dimensions, it has served to inspire many of the current research efforts in speech understanding.

2.4.2 Carnegie-Mellon's Systems

A speech understanding system has been developed at Carnegie-Mellon University [66-68, 56] for understanding chess moves (in the context of a real game so that semantic support can come from a chess-playing program). The context-free grammar is small (18 rules) and is capable of generating only a finite number of sentences

(about 5 million) using a 31 word vocabulary. The role of the parser is to predict the next element of input, not to build a syntactic structure for the utterance. It is mostly bottom up but operates in a top down mode to verify or reject hypothesies made by other components of the system.

This system is not syntax-driven, that is, syntactic analyses and predictions do not necessarily take precedence over other independent processes such as semantics, but it uses syntax in conjunction with semantics and acoustics to make decisions about the content of the utterance.

It operates in 5-10 times real time with about 99% word accuracy. It has been expanded to other data bases, but without fundamental change in scope or complexity of the grammar. (Although without comparable semantics, the performance in these other domains is relatively poor.)

Another speech understanding system, called DRAGON [4,5], has been implemented at Carnegie-Mellon. DRAGON models knowledge sources (acoustic-phonetic, lexical, phonological, syntactic-semantic) as probabilistic functions of Markov processes and uses an optimal search strategy to, in effect, search all possible sentences allowed by the (finite state) grammar, all pronunciations

of each sentence, and all matches of the phonetic strings to the acoustic observations in order to arrive at the best intepretation. When tested on 102 utterances with vocabularies ranging from 24 to 194 words, 49% of the utterances (and 83% of the 578 words) were correctly recognized in 48 to 174 times real time.

2.4.3 SDC's System

A speech understanding system at SDC is designed to handle queries and commands to a data management system with a data base of information about submarine fleets [6, 69]. Examples of typical input are "Print manufacturer where product equals automobile" and "Total quantity where type equals nuclear." When such unnatural sentences are spoken by humans the words tend to be pronounced almost as if they were in isolation rather than in the continuum one expects from connected speech, thus making the problem of acoustic analysis much easier.

The system is basically syntax-driven and uses a small context free grammar. Semantic constraints are built into the "syntactic" categories of the grammar. The system does not require that the utterance be processed strictly from left to right, but there is no bottom up processing from the word level (initial words are predicted by the discourse level controller). Syntax then

predicts words to either side which acoustics must verify.

Preliminary statistics indicate a 52% comprehension rate with a vocabulary of 150 words and a grammar of 35 rules. The strict context freeness of the grammar is critical, however, and it would be difficult to extend the system to allow more natural input.

2.4.4 SRI's System

A syntax-driven system to understand speech in the domain of a person requesting help with repairing a mechanical object such as a leaky faucet is under way at Stanford Research Institute [59, 60, 84, 85, 87]. The system operates from left to right through the utterance using a top down, highly predictive syntactic component which is aided in making predictions of the next word by semantic restrictions embedded in the parsing procedure. The acoustic component is used only to verify the predictions.

In this case, the grammar and parser are not separable but are represented together as a program whose execution results in a parse. The parser is not strictly depth first, but is "best first" in the sense that alternative parse paths are scored, the best one is extended one step and re-scored, and the process is

repeated. There is some effort made to minimize re-parsing due to altering one or more words of input by remembering successful paths previously encountered and following them again as far as possible.

Since an acoustic word matching function must be written for every word in the vocabulary, the current vocabulary of the system is effectively 54 words although the parser could handle up to 300. When the system was tested with 71 utterances, it responded to 51; 86% of the responses were correct. It required on the order of 200 times real time for the analysis.

The system's principal limitations are its strict left to right approach and the lack of bottom up techniques. This means that the syntactic and semantic expectations influence the direction of the analysis more than the acoustic component, so that no advantage can be taken of really robust acoustic information.

2.4.5 Lincoln's System

A speech understanding system was developed at MIT Lincoln Laboratory (see Forgie [24-26]) using the task domain of a retrieval, analysis, and display system for acoustic-phonetic data. Several linguistic processing modules have been written for the system (see [23-25]).

The primary one, called VASSAL, uses a context free grammar with 111 rules which are capable of generating only a finite language. Semantic information is embedded in the grammar by having the nonterminal symbols of the grammar be semantic rather than syntactic categories. Using a 250 word vocabulary, the linguistic processor operates top down, left to right through the utterance to hypothesize word strings to be compared against the acoustic analysis of the utterance.

2.4.6 LPARS

LPARS, a locally organized parser for spoken input written by Miller [53], is not a complete speech understanding system because it lacks an acoustic front end, but it is discussed here because it claims to provide all the higher level processing needed for such a system.

A program to accept the correct phonemic description of an utterance and output a scrambled version of it by making random substitutions and deletions is used to simulate an acoustic front end for LPARS. Unfortunately, the scrambler never inserts spurious phonemes as a real acoustic processor is likely to do, nor does it induce any fuzziness in the input in the sense of assigning more than one possible phoneme to a segment. A word matching program then matches the phonetic spelling of each word in

the dictionary at each position in the input to determine how likely the word is to occur there. When a word which matches above a given threshold is found in the input, a process to predict other words which may occur nearby is invoked, but this is very ad hoc and uses no general semantic or even syntactic guidance. Using a modified transition network grammar as a guide, LPARS then constructs, bottom up, all possible syntactic paths through each group of adjacent words. The system then syntactically proposes words or word strings to fill the gaps.

Unfortunately the form of the grammar is so restricted that it no longer possesses the full capability of an ATN but is really closer to context free. It includes a mechanism to check semantic feature agreement between parts of the sentence but does not have general register setting and checking capabilities. It handles only regular verbs, and only the present tense so that affixes and auxiliary verbs are minimized. Only singular NP's are permitted, so no number agreement checks need be made. Since general register setting and structure building actions are prohibited, the parser finds only a surface structure parse for the utterance, not a deep structure.

The reduction of the combinatorial possibilities was achieved not by clever heuristics or merging of representations but by legislating out troublesome data and input. For example, every word has only one part of speech in the lexicon, and for purposes of word matching and proposal, clusters such as "on the left of" are considered to be one word.

The system works on a limited vocabulary (72 words, most of them quite long) and a small grammar (24 states, 31 arcs). Because it processes, bottom up, all possibilities in parallel it would explode if given a large grammar or vocabulary. It would also be difficult to modify the system to do the actions on arcs characteristic of a ATN.

2.4.7 BBN SPEECHLIS

The speech understanding system under development at Bolt Beranek and Newman Inc. [7, 12, 55, 70, 70, 75, 94, 96] has used two task domains; that of the LUNAR text question-answering system [97] which deals with chemical analyses of Apollo 11 moon rocks and one dealing with travel budget management.

The overall design of the system is illustrated in Figure 2.4. There are seven basic components of the system. The acoustics component analyzes the acoustic signal to extract features and segment the utterance into a lattice of alternative possible sequences of phonemes (Schwartz and Makhoul [75]), phonological rules augment the output of the acoustic component to include sequences of phonemes which could have resulted in the observed phonemes; the lexical retrieval component retrieves words from the lexicon on the basis of this information (Rovner, et.al. [71]); the word matcher determines the degree to which the ideal phonetic spelling of a given word matches the acoustic analysis at a particular location [71]. All of these components structure their output in such a way as to represent the ambiguity which is inherent in their analyses. For example, they can be used to produce word lattices such as that which was shown in Figure 1.1.

The syntactic component is SPARSER, the system comprising the body of this thesis (see also Bates [7]). Acceptable utterances are not restricted to context-free syntax, since the grammar which SPARSER uses is a modified ATN grammar, capable of handling a large, natural subset of English. The remaining chapters of this thesis detail the structure and operation of SPARSER.

The semantic component uses a semantic network to associate semantically related words and to judge the meaningfulness of a hypothesized interpretation (See Nash-Webber [55]). This semantic formalism is much more general than any of the previously discussed systems, although a new network must be constructed for each new task domain.

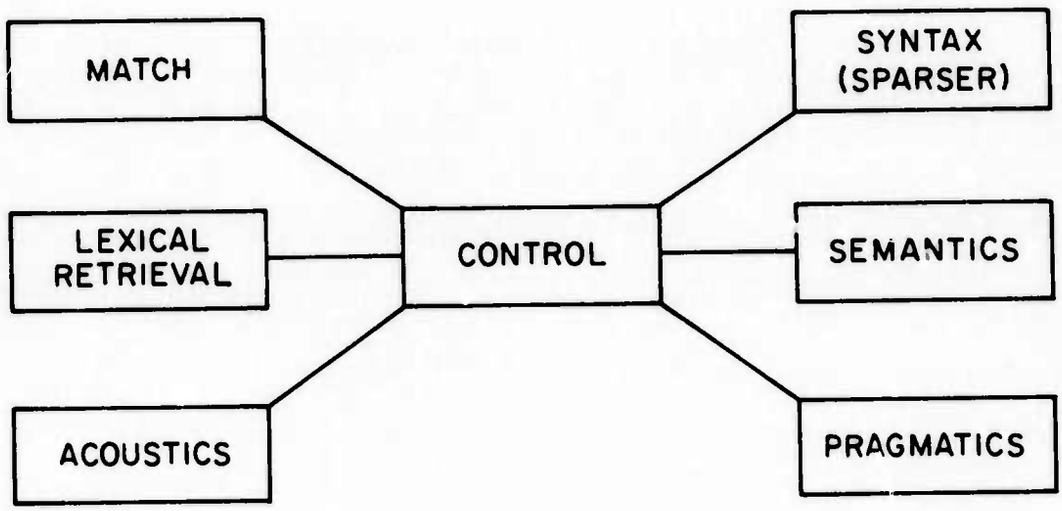


Figure 2.4

Design of BBN SPEECHLIS

The pragmatics component is not yet implemented, but is projected to contain information about the past dialogue, a model of the user, and other pragmatic data (see Bruce [12]).

A control component contains an overall strategy for employing the other components in order to obtain an interpretation of an utterance (see Rovner, et.al. [70]). It decides which component is to be called, what input it is to be given, and what is to be done with the output. It sets thresholds on word match quality. It combines the scores produced by the other components in order to rank competing hypothesies, and is the primary interface to all other components.

2.5 CONCLUSIONS

All but two of the speech understanding systems described above are syntax-driven. This is understandable because syntax is the best understood and most formalizable aspect of language analysis. However, that does not mean that it should provide overall guidance for the entire understanding effort.

When people speak naturally and informally they frequently make grammatical mistakes and seldom make semantic ones, yet they are easily understood. An automatic system driven by a syntactic processor which could handle only syntactically correct input would fail to understand such utterances, and would likely be difficult to modify to accept them. A system in wnich syntax was one of a number of equally important components contributing to the understanding of a sentence should be able to do a better job, since a bad report from syntax could be overridden by acoustics, semantics, or some other source of knowledge. It should be noted that none of the systems just described attempt to deal with ungrammatical utterances at the current time.

With one exception, all the speech understanding systems described above separate the parser from the grammar which drives it. This allows grammars which have been previously developed for text processing to be used, though modifications are usually necessary. The grammar can then be written in a meta-language which is convenient to the task and which is as different from the language of the parser which processes it as a high-level programming language is from machine code. It also allows a potential user to learn to construct grammars without having to learn to write programs -- a distinct advantage in

motivating linguists without programming experience to contribute to the development of the system. It also reflects the principle (held by some but not all linguists) that a grammar should be a neutral description of a language, biased as little as possible toward generation or analysis, thus maintaining the potential for the same grammar to eventually be used in a system to both understand spoken input and produce natural language output. The separation of parser from grammar also allows the grammar to be examined and modified without risk of introducing bugs into the parser, it permits various parsing processes to be developed for the same grammar, and it usually permits easy experimentation with various control structures within the parser.

With the exception of BBN SPEECHLIS, all of the systems above work with either a small vocabulary (which lessens the degree of lexical ambiguity in the input) or a limited grammar, or both, and so do not deal effectively with the combinatorial explosion of syntactic possibilities which would arise given a large (>1000 words) vocabulary and a reasonably general grammar.

What is needed in the syntactic component of a speech understanding system is a scheme which can:
(a) blend top down techniques with bottom up ones to combine directed predictive analysis with immunity to

errors in non-local context

(b) merge common information on alternate paths as much as possible to avoid re-parsing and facilitate decision making

(c) pursue most likely paths first while holding alternatives for further processing if necessary

(d) take advantage of the constraints which may be supplied by other sources of knowledge, such as acoustics, semantics, pragmatics, and prosodics.

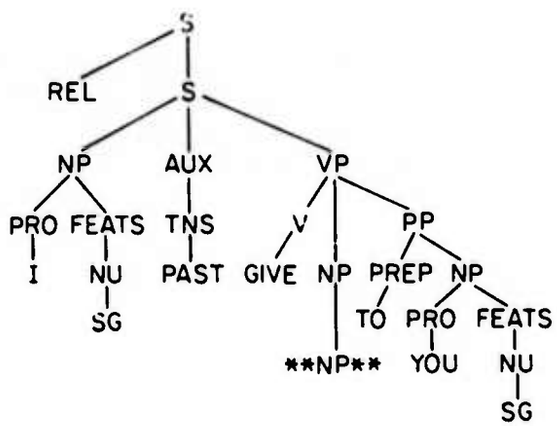


Figure 2.5

Chapter 3

The Grammar

3.1 INTRODUCTION

We have chosen the Augmented Transition Network formalism for the grammar which drives SPARSER because, as was mentioned in Section 2.3.4, it is a representation which allows merging of common portions of the analysis, it is amenable to both bottom up and top down parsing techniques, it fairly clearly separates the use of local information from information which was obtained from a distant portion of the utterance and, the author's previous experience with a large ATN grammar for parsing text laid the groundwork for the development of a similar grammar for speech.

We have tried as much as possible to keep the formalism which was developed by Woods [91-93] intact, but some changes have been necessary. The following section describes Woods' original formalism, concentrating on those areas which have been changed for the speech grammar. It assumes most of the information presented in Section 2.3.4 and it may be skipped by readers already familiar with ATNs. Section 3.2 describes the

modifications which have been made to Woods' formalism, and the final section describes the pre-processing which is performed on the grammer so that the speech parser can use it.

3.2 AUGMENTED TRANSITION NETWORK GRAMMARS

The concept of an Augmented Transition Network (ATN) Grammar was introduced briefly in Section 2.3.4 as a finite state network which has been extended to allow recursion, tests on arcs, and structure-building actions on arcs. The sequence of arcs which is taken during parsing reflects the surface structure of the input string, but the actions allow the creation of a deep structure which may be quite different from the surface structure.

The form of an ATN grammar is as follows. Each state of the network has a unique name. Associated with each state is an ordered list of arcs. There are eight types of arcs, which follow the schemas below. (An arc is a list of elements which is enclosed in parenthesies. An element may be a single word or another list. Capitalized words are actual elements, lower case words in brackets are descriptions of elements which will be described below, and * is the Kleene star operator which indicates

zero or more occurrences of the previous element.)

(CAT <category> <test> <action>* (TO <nextstate>))
(WRD <word> <test> <action>* (TO <nextstate>))
(MEM <list> <test> <action>* (TO <nextstate>))
(TST <label> <test> <action>* (TO <nextstate>))
(PUSH <state> <test> <action>* (TO <nextstate>))
(VIR <category> <test> <action>* (TO <nextstate>))
(JUMP <nextstate> <test> <action>*)
(POP <form> <test>)

The first element of each arc indicates its type. The interpretation of the second element depends on the type of the arc and will be explained below. The third element is an arbitrary test which must be satisfied in order for the arc to be taken. Actions, which may occur in any number on all arcs but POP arcs, generally manipulate information that is stored in registers. (A register is like a temporary variable which may contain a value.) The register contents are constants (flags) or pieces of syntactic structure which are built using previous register contents and/or the current item of input and/or the features of the current item. The last element of every arc type except JUMP and POP indicates which state of the grammar is to be considered next.

A CAT arc may be taken if the current input word is of the syntactic category specified by the second element of the arc. A WRD arc specifies the exact word which is required, rather than a syntactic category, and a MEM arc is exactly like a WRD arc except that the input word must be one of a list of words specified on the arc.

A TST arc performs just the test to determine whether the arc may be taken; the second element of this arc is a dummy label which is never used by the parser. A JUMP arc specifies the state to which a jump transition is to be made without "consuming" anything from the input string. Notice that TST and JUMP arcs are very similar, but the former is intended to test some feature of the input and move the input pointer over it while the later performs a similar test but does not move the pointer.

A VIR arc checks to see whether a constituent of the named category has been placed on the HOLD list by a HOLD action of some previous arc (see below). A PUSH arc initiates a recursive call to the network, beginning in the indicated state, to look for a constituent. A POP arc, which has no destination state, marks the state which it leaves as a terminal state for some level of the network; it also indicates the form (syntactic structure) which is to be returned as the result of the analysis of that portion of input parsed by the current level.

When the parser is operating, a number of registers are active. Whenever a PUSH occurs, this register list, along with other information, is saved on a stack while the parser recursively operates on the new (lower) level beginning with an empty register list. When a POP arc is taken, the stack is popped, wiping out the current register list and restoring the register list which was current before the last PUSH. The constituent which was POPed then becomes the current input item for the PUSH arc.

Several types of actions may occur on the arcs. The most common is (SETR <reg> <form>) which sets the indicated register to the value of the form. The action (SENDR <reg> <form> which may be used on a PUSH arc, causes the register to be set to the value of the form at the lower level of recursion about to be initiated by the PUSH, i.e. the register value will initialize the register list after the PUSH. (LIFTR <reg> <form> <where>) is the inverse of SENDR in that it sets the register to the value of the form at the level specified by the <where> form. The <where> specification can be an integer indicating the number of levels up or it can be a predicate, in which case the nearest level at which the predicate is true is chosen. Thus LIFTR and SENDR are used to communicate between levels of recursion; an

analysis may send information from part of one constituent into the process which parses a sub-constituent, and information which is not necessarily part of the structure of a constituent can be given to the process which will continue after the parsing of the constituent is complete.

For example, when parsing the noun phrase "the person who travels to Washington" the grammar may PUSH for a relative clause following "person". It is necessary, at some point, to check for number agreement between the noun "person" and the verb "travels" since "* the people who travels to Washington" must not be allowed (and of course must be rejected by a speech parser as early as possible). SENDR can be used to send down the head of the noun phrase to the relative clause level so that the agreement test can be made as soon as the verb is encountered.

The action (HOLD <constit>) places the indicated constituent on the HOLD list, a global variable which is accessible at all levels. This action together with the VIR arc constitute a mechanism for dealing with the phenomenon called left extraposition in transformational grammar theory, which moves a sub-constituent from an embedded constituent up and to the left of its deep structure position. An examples of this is the fronting of questioned noun phrases ("Which doctor did he send you to?"). Since the extracted constituent can be moved only

to portions of the deep structure which dominate its original position, once an item has been put on the HOLD list it must be removed (by using a VIR arc) before a POP is done from the level at which it was placed. However the VIR arc which picks it up may occur many levels below where it was found.

The actions RESUMETAG and RESUME are used to handle the structures produced in transformational grammar theory by the transformation known as right extraposition. For example, the sentence "The place I went to which was on the West Coast" has a deep structure constituent for "the place which was on the West Coast" but the relative clause has been moved from its original position in the founted noun phrase to the right in the surface structure.

The function (RESUMETAG state), when used as an action on an arc, creates a marker combining the named state (at which parsing could continue at some later time) with the current register list. Then at some later time, if the action (RESUME) is encountered on an arc the marker is retrieved and the configuration the parser was in when the RESUMETAG was executed is re-established. The extraposed text can be parsed as usual, and when a POP is done the completed constituent is used for the input item on the arc containing the RESUME.

3.3 MODIFICATIONS

A number of modifications have been made to the ATN formalism just described in order to make it more amenable to use by a parser for speech. None of these changes reduce the power of the grammar, and some, as will be shown, may even clarify some points. We call the resulting formalism a Modified Augmented Transition Network (MATN).

3.3.1 Tests on the Arcs

Every arc of an ordinary ATN has a test component, which may be any predicate. It is usually a boolean combination of tests on the current input word (its features, etc.) and the contents of registers which have been set by actions on previous arcs. In the MATN formalism, the test component of each arc is, on all but the PUSH arc, a list of two tests. The first is a test on the current word and its features, i.e. a local, context-free test. The second is a test on the register contents, i.e. a context-sensitive test. Both tests must succeed for the arc to be taken. In the rare case where a disjunction of current word information and register contents is required, it may appear in the context sensitive slot. (Experience with the LUNAR grammar [97] and the speech grammar has shown, however, that such tests

are rarely used.)

The reason for splitting up the tests in this way is that register checking tests cannot be made unless the registers are set, and in many situations in the speech environment there may not be enough left context to guarantee that the proper registers would be set. Thus it is useful (as will be seen in the next chapter) to be able to evaluate the context-free test on an arc at a different time in the parsing process from the context-sensitive one.

On PUSH arcs, there are three types of tests which are used. It is useful and efficient to test the next word of input before actually doing the PUSH, to see, for example, if the next word can begin a constituent of the type being PUSHed for. This test is called a look-ahead test, and takes the place of the normal context-free test in the test component of the arc. There is also the usual context-sensitive test on registers which were set before the PUSH arc was encountered. And finally, when the PUSH arc returns with a constituent, another context-free test may be done on the structure of the entire constituent. Therefore, the test component of a PUSH arc is a list of the three tests just described.

3.3.2 LIFTR's

The action LIFTR has been removed from MATN grammars, but the ability to send information to a higher level has been preserved in another form.

There is a special register called LIFTLIST which may be set by the normal SETR action. When a constituent is POPPed, the contents of the LIFTLIST register is attached to the constituent in the WFST in such a way that it becomes the features list on the PUSH arc which picks up the constituent. Actions on the PUSH arc can then access the features list in any way and manipulate the information there, for example, to attach it to the LIFTLIST register at this level so it will be passed up again.

Originally LIFTLIST was conceived of as a way to attach features to a constituent which did not really have a place in the syntactic structure. For example, one might want to pop a number as the structure (NP N NUMBER 105) with the feature DIGITS to indicate that it had been parsed from "one oh five" instead of "a hundred and five". However, LIFTLIST can be generalized to a mechanism for passing any information up to the higher level, as long as the actions on the PUSH arc can interpret the information.

3.3.3 SENDR's

SENDR's were an efficient mechanism for text parsing because they allowed tests to be made on a lower level which involved information obtained somewhere (possibly far) to the left in the input string -- information which would normally be inaccessible because it would be hidden on the stack during the parsing of sub-constituents.

There are several reasons for not allowing this mechanism in the speech parser. Suppose, in the input that looks like "... the person who travels ...", the word "person" is not the word which was really uttered. If it were allowed to be passed down it would become an integral part of the analysis at the lower level, and if another word were to be hypothesized in its place, the lower level the analysis would have to be redone even if none of the words in the relative clause had been changed. This is a process which would be extremely wasteful, especially in the speech environment where one wants to be able to take as much advantage as possible of information which was gained at one point and slightly altered at another. In particular, it is advantageous to consider as constituents such constructions as relative clauses so that they can be placed in a well-formed-substring table for use by other processes.

Another reason is that some types of verifications (semantic, prosodic, and pragmatic, at least) can be done most conveniently on portions of an utterance which have been assigned a syntactic structure, i.e. on constituents. If a portion of an utterance is parsed (e.g. "that I gave you" from the complete utterance "The book that I gave you") but does not form a complete constituent because it is missing a piece of information from a higher constituent to the left which would have been sent down had it been available, then these verifications may not be made until the missing word or words are identified. Yet it may be important to build and verify the constituent in order to predict the missing word to the left. Therefore, it is better to allow constituents to be built without information which would normally have been passed down. When parsing possibly incorrect fragments with little or no left context, it is best to keep constituents as small and as independent as possible.

Since SENDR's are not permited in the MATN grammars, it is reasonable to investigete the problem of converting a regular ATN grammar to MATN formalism in this respect. (Of course, starting over from scratch to build a new grammar is also an acceptable, and perhaps a more reasonable approach, but for purposes of comparison a

method of conversion will be discussed.) We will first discuss a general method for performing the conversion, and then show that in some cases it can be considerably simplified.

Instead of sending down information on a PUSH arc to state X, the PUSH is made to a new state Y which then has a JUMP arc to state X. On this JUMP arc, the registers which would have been sent down are set. They may be set to a constant value if they are merely flags (e.g. (SENDR TYPE (QUOTE REL)) can be replaced by (SETR TYPE (QUOTE REL)) at the lower level). If a register was meant to contain other information, it is set to a distinctive dummy symbol (e.g. (SENDR SUBJ (GETR NP)) can be replaced by (SETR SUBJ (QUOTE **NP**))). The PUSH arc which originally contained the SENDR's must be changed beyond removing them and replacing the state; it must have an action which will take the constituent returned, do any agreement checks which are necessary (aborting the arc if the check fails) and perhaps replace the dummy node by the appropriate structure. The structure returned by the PUSH for a relative clause on the fragment "that I gave you" might look like Figure 3.1 (where the structure is shown in both the usual tree diagram form and a corresponding form more amenable to computer output).

```
S REL
  S NP PRO I
       FEATS NU SG
    AUX TNS PAST
    VP V GIVE
       NP **NP**
       PP PREP TO
        NP PRO YOU
             FEATS NU SG
```

Figure 3.1

Two Representations of a Parse Tree

The PUSH arc may or may not replace the dummy node after making the necessary tests. In addition, tests on the lower level must know about the dummy nodes so that they will accept them.

This procedure may seem excessively complicated, but is actually quite straightforward. Because it is not always necessary to create a new state and arc every time a PUSH arc is modified (since in practice many of the registers sent down are constants -- nearly 40% of the SENDR's in the LUNAR grammar were of this type), the size of the grammar increases only slightly. The resulting grammar is still clear and compact, and may even give a more explicit picture of the parsing process than the original, since the network for each constituent is self

contained and has no references to registers set elsewhere in the grammar.

3.3.4 The HOLD List

The HOLD list, along with HOLD actions and VIR arcs, has been eliminated from MATN grammars. The HOLD list can be replaced in ordinary ATN grammars by using SENDR to send down items being held every time a PUSH is done and testing for that item in place of a VIR arc. Since the SENDR can also be removed by the process described above, the HOLD list can be eliminated in the same way.

It is not really necessary to go through this two level process in order to eliminate HOLD's from an ATN. In many cases, instead of holding a constituent to be picked up at some lower level, the grammar may simply put the constituent in a register and then after some lower level constituent has been parsed, embed the held item in the proper place. This eliminates the use of a dummy register, and is usually feasible.

3.3.5 RESUMETAG and RESUME

Several things may be noted about right extraposition which can be used to simplify the way it is handled. Firstly, the portion of the sentence which is extraposed

must be a constituent, such as a relative clause or a prepositional phrase. Secondly, the portion of the constituent which is left behind when such a sub-constituent is extraposed still forms a complete constituent. Thus we can have "How many samples were there which were analyzed" or "How many were there which were analyzed" where "how many" can be considered a noun phrase with an elided noun, but we cannot have *"How many were there samples which were analyzed." Finally, the constituent which is extraposed is always moved up, out of the original constituent.

The function (RESUMETAG state) is still allowed in MATN grammars, but it has a slightly modified effect. Using the LIFTLIST mechanism, it performs the equivalent of (LIFTR RESUMESTATE state) and (LIFTR RESUMEREGS REGS), i.e. on the next higher level it sets the register RESUMESTATE to the named state and the register RESUMEREGS to the register list which was current just before RESUMETAG was called.

Instead of calling the function RESUME on some later arc, the MATN grammar writer must have a PUSH arc which PUSHes to the state named by the RESUMETAG and has a test to guarantee that the PUSH is done only if the register RESUMESTATE is set to the same state to which the PUSH is to be made. The actions on the PUSH arc can then use the

register information stored in RESUMEREGS together with the constituent returned from the PUSH to build the entire deep structure constituent.

By eliminating the RESUME action and requiring a PUSH arc with a confirming test, we have required a little more work from the grammar writer, but once the grammar is written it is easier to see what is happening in the parsing process. It also simplifies the automatic indexing of the grammar (see Section 3.4) since the state in which the parsing is to resume is explicitly named in the grammar at the point it is to be used, rather than held in a register which is available only during the actual parsing process.

3.3.6 Weights on Arcs

The fourth element of every arc in a MATN is a small integer which is called the weight of the arc. This weight was originally conceived of as a rough measure of either (a) how likely the arc is to be taken when the parser is in that state or (b) how much information is likely to be gained from taking this arc, i.e. whether the parse path will block quickly if the arc is wrong. That these two schemes are not equivalent can be seen by the following example. In a given state, say just after the main verb of the sentence has been found, the arc

which accepts a particle may be much less likely than the arc which jumps to another state to look for complements. However if a particle which agrees with the verb is found in the input stream at this point, then the particle arc is more likely to be correct.

Since it is not at all clear how to measure or even intuit how much information is likely to be gained from taking an arc, it was decided that the weights would reflect relative likelihoods. The actual weights which have been used in the speech grammar reflect an intuitive, though experienced guess as to how likely the arc is to be correct if it is taken, assuming the state itself is on the correct path.

3.3.7 Minor Changes

Several minor changes have been made to allow more concise representation of information in the grammar.

First, MEM arcs have been eliminated and their function taken over by allowing WRD arcs to have a list of words as the second element. In cases where this list is long, especially if it is to be used at several points in the grammar, an indirect pointer to the list may be used. That is, the second element of a WRD arc may consist of an atom surrounded by slashes (/'s), provided that the

slashed atom is a global variable which is bound to the list of words which would normally appear in this position on the arc. Thus arcs of the form (WRD (WHICH THAT) ...) or (WRD /MONTH/ ...) where /MONTH/ = (JANUARY ... DECEMBER) are allowed.

Another minor change is the inclusion of actions on POP arcs. This is useful to allow the LIFTLIST register to be set just before the POP occurs. If actions were not permitted on the POP arc, they might have to be duplicated on each arc entering the final state.

3.3.8 Summary and Sample Grammars

The form of the arcs of a MATN grammar are:

```
(CAT <category> (<cftest> <cstest>) <weight> <action>*
                                        (TO <nextstate>))
(WRD <word>|<list>|<pointer> (<cftest> <cstest>) <weight>
                        <action>* (TO <nextstate>))
(TST <label> (<cftest> <cstest>) <weight> <action>*
                                        (TO <nextstate>))
(JUMP <nextstate> (<cftest> <cstest>) <weight>
                                        <action>*)
(PUSH <state> (<look-ahead> <cstest> <constittest>)
                <weight> <action>* (TO <nextstate>))
(POP <form> (<cftest> <cstest>) <weight> <action>*)
```

Two grammars which will figure predominantly in the remainder of this thesis have been written in the MATN formalism. One is an extensive grammar which can handle many questions, declaratives, noun phrase utterances, imperatives, active and passive forms, relative clauses (reduced and unreduced), complements, simple quantifiers, noun-noun modifiers, verb-particle constructions, numbers, and dates (but not conjunctions). It began as a modification of the grammar for the LUNAR system [97] but has been considerably adapted and expanded. This grammar is called SPEECHGRAMMAR, and is listed in Appendix II. Many of the examples in Chapter Six were produced using this grammar.

For some illustrative purposes, SPEECHGRAMMAR is too big and complex, so we have produced a MINIGRAMMAR which will be used to show the basic operation of the speech parser. A detailed listing is given in Appendix I, but the diagram in Figure 3.2 probably shows the structure more clearly. Two copies of Figure 3.2 are given, so that one may be torn out for easy reference when reading the following two chapters.

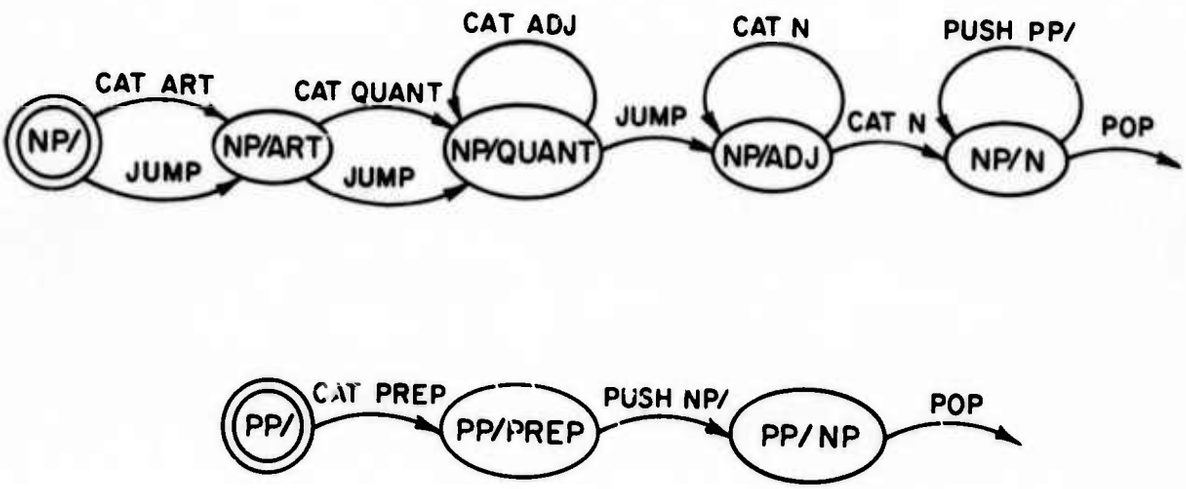


Figure 3.2

MINIGRAMMAR

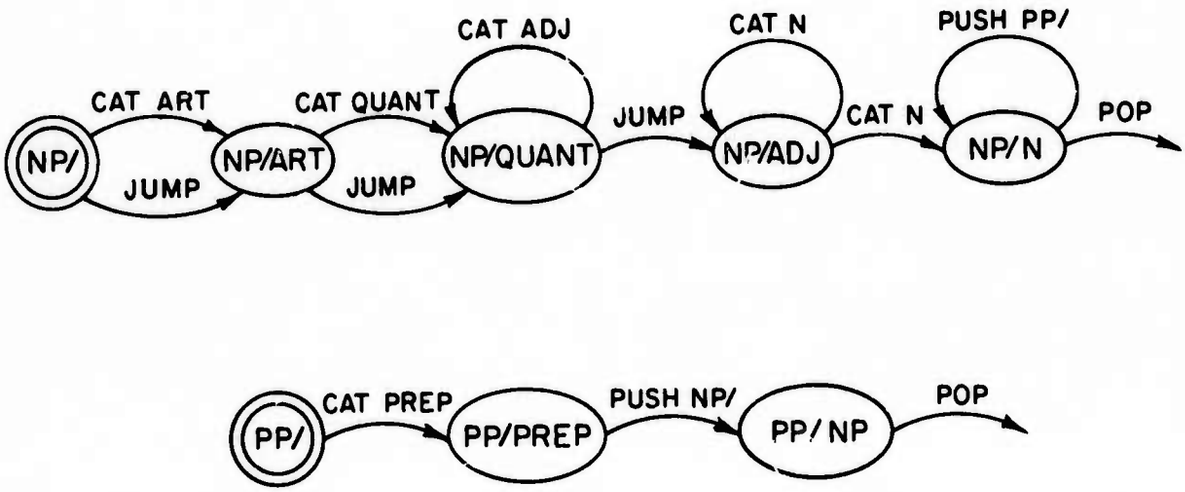


Figure 3.2

MINIGRAMMAR

3.4 THE GRAMMAR INDEX

When an ATN parser begins to parse a sentence, it knows that it has to begin in the initial state of the grammar with the first word of the sentence. A speech parser, however, may not be able to parse left to right through the utterance. This implies that given a word somewhere in the middle of an utterance the parser must be able to figure out the state (or states) in which to begin. Then in order to move from right to left (to predict what could precede that first given word) the parser must be able to determine for any state which arcs can enter it, and for any arc which state it comes from. Since the grammar is organized for normal parsing in just the opposite fashion, i.e. for any state one can determine what arcs leave it and for any arc (except POP) one can determine which state it terminates on, it is necessary to build an index into the grammar.

This index consists of a number of tables which contain pre-computed information about a number of aspects of the grammar. For example, one table associates with each state of the grammar the list of pop arcs which can be reached on that level. When accessing information "backward" in the grammar, i.e. looking for an arc which has some property relative to the state on which it terminates, it is always necessary to know the state from

which the arc originated in order to extend the parse path from right to left, so in all such cases not just the arc alone but a state/arc pair is returned. Such a state/arc pair is hereafter called a starc and is endowed with the properties of the arc it contains (so that we can talk about a starc which PUSHes or JUMPs, for example).

The indexing function operates by walking recursively through the grammar, filling in various tables as it goes. For example, if A is a PUSH arc from state S1 which terminates on state S2 and PUSHes to state S3, then the following actions are performed:

1) The starc (S1 A) is associated with the entry S2 in a table which shows for every state the starcs with PUSH arcs which terminate there.

2) The starc (S1 A) is associated with the entry S3 in a table which shows for every state the starcs which can PUSH to it.

3) The state S3 is added to the global list PUSHSTATES, if it is not already there. (The parser can use this list when processing right to left to determine when it has come to the beginning of a constituent.)

Once these tables are set up, they are used to produce another table which shows for any state all the possible paths (sequences of starcs) which can terminate on that state without using the previous word of input.

The utility of this table will be shown in the next chapter.

The retrieval functions which return the pre-computed values stored in the tables are the following:

(ARCSJUMPTOS state) -- returns a list of starcs which jump to the given state without looking at the current input.

(ARCSJUMP*TOS state), -- returns a list of starcs which jump to the given state and perform some test on the input on the way.

(ARCSPUSHTOS state) which returns a list of the starcs which can PUSH to the given state.

(ARCPOPSTOS arc) -- for a POP arc, returns the list of states which terminate PUSH arcs where there is a path from the state PUSHed to to the POP arc given.

(ARCSPUSHJUMPTOS state) -- returns the starcs with PUSH arcs which terminate on the given state.

(ARCSTOS state) -- returns the list of input-using starcs which terminate on the given state.

(ARCSUSING wrd/cat) -- returns the list of WRD or CAT starcs which will accept the given word or syntactic category.

(STATESSTARTPUSH state) -- returns the list of states which have PUSH arcs to the given state.

(POPSTOS arc) -- returns the list of states which

terminate PUSH arcs, where the lower level path initiated by the PUSH can end with the given POP arc.

The following chapter will show how these retrieval functions are used to guide the parser and to make syntactic predictions to fill in gaps in the utterance.

Chapter 4

Overview of SPARSER

4.1 INTRODUCTION

SPARSER is composed of two main parts, a parser and a grammar. The form of the grammar has already been discussed in Chapter Three. This chapter will outline the operation of the parser and will describe the data structures it constructs and uses.

4.2 INPUT

The input to the parser is assumed to be a set of words together with their boundary points (which may or may not be related to points in time). A word together with its boundaries is termed a word match. A word match also includes a score which indicates how well the ideal phonemic representation of the word matched the acoustic analysis of the utterance (but as we shall see the parser has little need of this information). Since the same word may match at several sets of boundary points or may match in several ways between the same boundary points, each word match is also given a unique number to help identify

it. Thus the structure for a basic word match is:

(number word leftboundary rightboundary lexicalscore)

e.g. (4 TRAVEL 5 11 94), or (4 TRAVEL 5 11 (94 110)) where the score is given as a pair of numbers representing the actual and maximum scores, or (4 TRAVEL 5 11) where the score is omitted.

How is the input to the parser to be constructed? We assume that acoustic processing and lexical scanning components can operate on a digitized waveform to produce a number of word matches such as previously shown in the word lattice of Figure 1.1. (That this is possible has been demonstrated by Woods [94]). Allowing the parser to operate unrestricted on the entire word lattice would probably not be fruitful because of the large number of locally syntactically correct combinations of words, but one possibility for input to the parser would be to take a set of the best-matching, non-overlapping word matches in the lattice, such as those in Figure 4.1.

A set of non-overlapping word matches is a hypothesis about the content of the utterance. In order to avoid creating large numbers of such sets which are put together combinatorially with no basis except local acoustic match, semantic or pragmatic processes can be used to group word matches based on what is meaningful or likely to be heard. For example, if a dialogue has been about various nickel

compounds, the combination "nickel analyses" may be more likely than "chemical analyses" even though the word match for "chemical" has a higher score than that for "nickel"". We will not attempt to detail here how this semantic grouping could be done and how the sets could be scored, since it has been described elsewhere [55].

```
DO MANY PEOPLE DONE CHEMICAL ANALYSES      ROCK
0  2     6       11   14       22        30  35  38

GIVE EIGHTY PEOPLE DONE  TEN  MODAL  DETERMINATION ROCK
0    3      6       11   14 15  18 21    26                35  38

WERE ANY PEOPLE       METAL          SEVEN
0    3   6      11     17    21       27   32
```

Figure 4.1

Sample Word Match Sets

Borrowing some more terminology from the BBN speech system, we will use the word <u>theory</u> to denote a set of word matches such as we have just described together with (possibly empty) slots for information from each of the possible knowledge sources in the system. From the point of view of SPARSER, usually only the word match portion of a theory is of interest, hence we shall fall into the habit of using the word "theory" to refer to the word match set it contains. When speaking of the syntactic

component of a theory, however, we are refering to the information slot for syntax which accompanies each word match set.

Theories have the following characteristics:

1) They contain a set of basic, non-overlapping word matches.

2) They tend at first to contain long content words and not many short function words. This is because long words are more reliably acoustically verified and content words are easier to relate semantically and pragmatically. Since small words such as "am", "do", "the", "one", "have", "of", "in", etc. may be represented by very little acoustic information, they would tend to match at many places in the utterance where they do not really occur. Consequently they are not searched for by the initial word match scan, nor are they proposed in the semantic stages of hypothesis formation.

3) They need not (and generally do not) completely span the utterance, but have numerous gaps of varying sizes (e.g. for the function words).

4) They tend to contain some sequences of contiguous word matches. Such a sequence is called an island.

That such a set of theories can be created has been demonstrated by the BBN SPEECHLIS system. The syntactic component, SPARSER, is expected to process these theories

one at a time. In certain circumstances which will be detailed later, the input to SPARSER will be a theory together with one or more word matches which are to be added in order to create a new larger theory which is then to be syntactically analyzed.

We will assume that there exists a control component like that described in Section 2.4.7 and in [70] which presents SPARSER with theories to process and to which SPARSER can communicate predictions and results.

4.3 OPERATION

4.3.1 Preliminaries

Given a theory, what is to be done with it? We begin by considering a subset of the question: Given an island of word matches, what is to be done with it? The answer is to create one or more parse paths through the island and to predict what words or syntactic classes could surround the island. A parse path is the sequence of arcs in the grammar which would be used by a conventional ATN parser to process the words in the island, if the island were embedded in a complete sentence.

For example, consider the way a parser might process an island of word matches such as (1 CHEMICAL 14 22) (2 ANALYSES 22 30) using the MINIGRAMMAR of the previous chapter. Beginning in state NP/ of the grammar (omitting for the moment the problem of how it is known that NP/ is the right place to begin) the sequence of arcs which would be taken to parse "chemical analyses" as a noun phrase is that shown below in Figure 4.2.

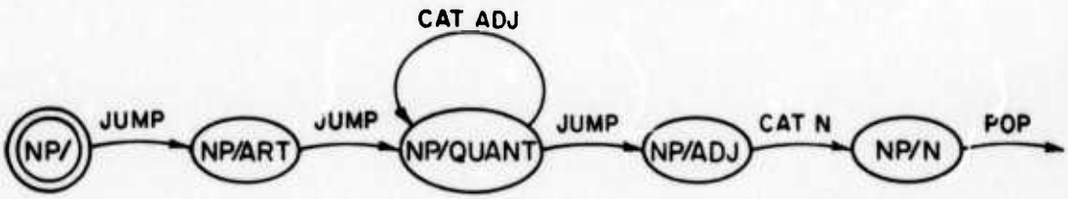


Figure 4.2

Portion of MINIGRAMMAR needed to parse "chemical analyses"

Let us define a <u>configuration</u> to be a representation of the parser being in a given state (say NP/QUANT) at a given point in the utterance (say 14). We will write configurations as STATE:POSITION in text (e.g. NP/QUANT:14) and schematically as a box within which are written the state and the position. If a configuration represents a state which is either the initial state of the grammar or a state which can be PUSHed to (i.e. a

state which can begin the parsing of a constituent), it is called an initial configuration, and is indicated schematically by a filled-in semi-circle attached to the left edge of the box. Note that a configuration NP/QUANT:14 is quite distinct from a configuration NP/QUANT:22 since they are at different positions in the input. In SPARSER, each configuration is also assigned a unique number which is a convenient internal pointer.

The process of traversing an arc of the grammar using a particular word is represented by a transition from one configuration to another. A transition can be made only if the arc type is compatible with the current item of input and if the context-free test on the arc is satisfied. (The context-sensitive tests are evaluated later.) A transition carries with it information about the arc which it represents and the item of input it uses. The item of input is usually the word match which the arc uses, but it is NIL in cases such as JUMP arcs which do not use input, and it is a complete constituent for PUSH arcs. A unique identifying number and the list of features, if any, which is associated with the input word or constituent are also recorded on the transition in SPARSER, but they are not shown schematically. A transition is represented schematically by an arrow from one configuration to another with an abbreviated form of

the arc written above the arrow and the item of input under it.

The syntactic part of any theory which SPARSER processes contains, among other things, lists of the transitions and configurations which are created or used by the theory. Thus when we talk about creating a configuration or transition it is implicitly understood that SPARSER also adds it to the appropriate list, and when we talk of adding an existing configuration or transition to a theory we mean adding it to the appropriate list. Therefore, removing a configuration or transition from a theory means removing it from the syntactic part of the theory, not removing it entirely from SPARSER's data base.

Like configurations, transitions are unique, so only one transition is ever constructed from point A to point B for arc X and input Y. We will frequently speak of creating a transition or a configuration, but the reader must bear in mind that if such a configuration or transition already exists, this fact will be recognized and the pre-existing configuration or transition will be used. (Timing measurements indicate that it takes about .052 seconds to create a configuration and only .01 seconds to test if a particular configuration already exists. For transitions, creation takes about .54 seconds

and recognition .012 seconds.

The sequence of configurations and transitions which would parse the above example is displayed in Figure 4.3.

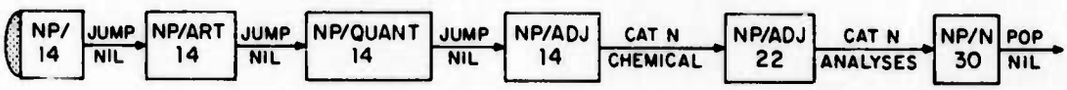


Figure 4.3

Path for parsing "chemical analyses"

A connected sequence of transitions and configurations is called a path. If the sequence begins with an initial configuration and ends with a transition representing a POP arc, it is a complete path, otherwise it is a partial path. Paths are assumed to be partial unless otherwise specified.

The following two sections describe how SPARSER creates paths such as that in Figure 4.3.

4.3.2 Beginning to Parse an Island

SPARSER processes an island of words by beginning with the leftmost word and determining its possible parts of speech. (This determination is currently made solely from a dictionary lookup but it could be modified by information provided by some other source such as semantics or pragmatics which knows, for example, that "budget" is more likely to be used as a noun than as a verb.) Then the arcs of the grammar which can process the word are found (by looking in the previously constructed index via the ARCSUSING function described in Section 3.4). For each arc, two configurations are constructed, one for the state at the tail of the arc and one for the state at the head, using the left and right boundary positions of the word match, respectively, and a transition for that arc using the current word match is also built. Schematically, we have for our example a situation which looks like that of Figure 4.4 (such a display of all or some of the transitions and configurations which the parser has constructed is called a map). Notice that a configuration may have any number of transitions entering or leaving it.

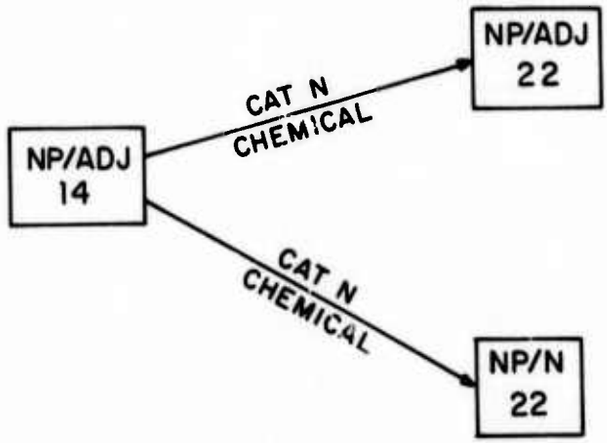


Figure 4.4

Initial map for parsing "chemical analyses"

The idea of this process is to begin to set up paths which may be used to parse the island. However it is not necessarily the case that the only configurations which could start paths through the island are those which have just been obtained, since it may be possible to create transitions which enter them via JUMP arcs or TST arcs. For each state, the sequence of arcs which can reach it without using the previous word of input have been be pre-calculated by the grammar indexing package and are retrievable by the functions ARCSJUMPTOS and ARCSJUMP*TOS, so the appropriate configurations and transitions may be constructed. These transitions are called lead-in transitions. Thus the map becomes that in Figure 4.5

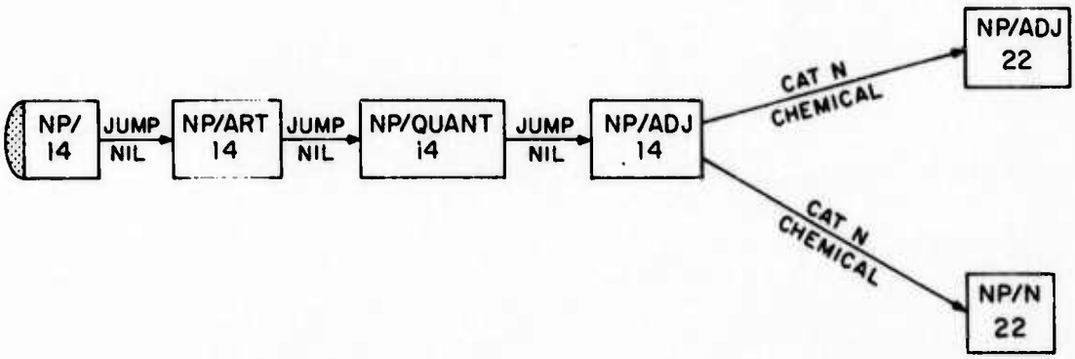


Figure 4.5

Lead-in transitions for parsing "chemical analyses"

Note that any of the configurations (except for NP/ADJ:22 and NP/N:22) could actually be the correct leftmost configuration for this island, depending upon what the (currently unknown) left context of the island is.

By looking in the grammar index, SPARSER can determine, for each configuration which could start the island, just what sort of left context could be appropriate. For example, the CAT ADJ arc in MINIGRAMMAR which enters state NP/QUANT implies that an adjective could precede the island and, if it did, the transition which would process it would terminate on configuration NP/ADJ:14.

Because the initial configuration NP/:14 could start the island, anything which could precede a noun phrase could occur to the left; again the grammar index provides

the information that the CAT PREP arc could lead to a configuration which could accept a noun phrase (via the PUSH NP/ arc), so a preposition could also prefix the island. If the index functions indicate that a constituent could be picked up by a PUSH arc which could terminate on the configuration under consideration, an indicatiion is made in the WFST so that any time a constituent of the desired type is built which ends at the proper location, it may be tried here.

Because of the highly recursive nature of ATN grammars, it is very likely that as we chain back through the possible sequences of PUSHes which could lead to the beginning of the current constituent (or the sequence of POPs which could be initiated by the completion of the current constituent) a large number of predictions will be made. Rather than make all these predictions automatically, before we are even sure that there is in fact a constituent at the current level, the possible configurations which could make predictions on other levels are saved to be activated later if the predictions from the current set of active configurations are not sufficient.

The predictions which are made (not saved) are not acted upon at this time, but are kept internally by SPARSER until all the islands of the theory have been

processed. We shall see in Section 4.3.5 what then becomes of the predictions.

4.3.3 Parsing Through an Island

Once processing has proceeded this far, we can go back and consider the set of configurations which represent states the parser could be in just after processing the first word of the island. In our example, these are configurations NP/ADJ:22 and NP/N:22. Configurations such as these which are waiting to be extended to the right are called active configurations. SPARSER selects a subset of the set of active configurations (how this subset is selected will be discussed in the next chapter) and for each configuration tries to extend it by trying to parse the rest of the island beginning in that configuration. When the parser is considering a configuration at some position, the input pointer is set to the word match of the island, if any, which begins at the same position in the input. If there is no such word match, i.e. at the end of an island, the input pointer is NIL.

The grammar associates with the state of the configuration a list of arcs which may be tested (using the arc type, the context free test on the arc, and the current input) to determine whether a transition can be

made to extend the path. We will consider each type of arc in turn, since the effects of taking various types of arcs are different, and explain for each case what happens if the arc is taken. Whether just one transition, or several, or all possible transitions are made from an active configuration is a matter to be discussed in Chapter Five.

A. Arcs which do not use the current input item.

Some JUMP arcs do not look at the current item, so they may be taken whether the input pointer is set to a word match or to NIL. The transition which results from taking an arc of this type has a null item associated with it, even if there is a word match in the theory at this point. The positions of the configurations at each end of the transition are the same; this corresponds to the fact that an ATN parser would not move the input pointer as a consequence of taking this arc.

B. Arcs which use the current item but do not consume it

Rarely, a JUMP arc may test the current item in some way, for example, to make a feature check. If there is no word match for input, an arc of this type cannot be taken. If there is a word match, it is noted on the transition which is created, but the configurations at each end of

the transition have the same position. (It is then the case that the next input-using or input-consuming transition on the path including this transition must use the same word match.)

C. Arcs which consume input

These are TST, CAT, and WRD arcs which end in a (TO nextstate) action. The operation is exactly the same as that in B above except that the configuration on which the transition terminates has the position of the right boundary of the current word match.

D. POP arcs

Taking a POP arc results in the creation of a transition which has a null item, because POP arcs are not permited to look at input. Although in Woods' original ATN formalism POP arc was a "pseudo" arc which merely indicated that the state to which it is attached could be considered a final state, here they are just as "real" as any other arc type and cause the creation of a transition which is just like that for other arc types, except that it has a NULL terminating configuration.

E. PUSH arcs

When a PUSH arc is encountered, a monitor is placed in the Well-Formed Substring Table (WFST) at the current position to await the occurrence of a constituent of the required type. If one or more such constituents are already in the table, then for each one there are three possibilities: it may be composed of word matches which are in the current theory, it may be composed of word matches some of which are not in the current theory but which could be added without violating the non-overlapping constraint, or it may be composed of word matches some of which are incompatible with the current theory.

In the first case a transition is set up using the constituent as the current word. The transition terminates on a configuration whose state is determined from the termination of the PUSH arc and whose position is that of the right boundary of the rightmost word match in the constituent.

In the second case, a notice is created and sent to the control component (see Sections 4.2 and 2.4.7). A notice is a request that SPARSER be called to enlarge a theory by adding some new information, in this case, some additional word matches which form a constituent that the theory can use. SPARSER does not try to determine when

(or even whether) the theory should be so enlarged. That is an issue for the main controller to decide (see Rovner, et.al. [70]). Section 4.3.7 will discuss how SPARSER enlarges a theory if called upon to do so.

In the final case, if there are no usable constituents in the WFST, a new configuration is set up to start looking for one and is added to the list of active configurations. Its state is the state specified by the PUSH arc and its position is the same as the current configuration.

There is a considerable amount of processing that can happen any time one of the transitions just discussed is made. Whenever an initial configuration is constructed, this fact is recorded in the configuration. Whenever a transition is made from such a configuration, the information that there is a path from some initial configuration is recorded on the subsequent configuration. Similarly, whenever a POP transition is made, the configuration it emanates from and all previous configurations on any path which can terminate with the POP transition are marked to indicate that they can reach a POP transition. Whenever a transition is made which completes a path from an initial configuration to a POP transition, the path is executed, one transition at a time, and the register setting actions and context

sensitive tests are executed. If a test fails or an arc aborts, the transitions and configurations of the path are removed from the list of configurations and transitions which are in the syntactic part of the current theory (unless they are used by another path in the theory) but not removed from the map. If the execution is successful, a deep structure tree is produced. That structure together with its features is given a score, which may include evaluations by other components such as semantics and prosodics, and is entered in the WFST.

It is quite important that sources of knowledge other than syntax be called upon to verify and to rank syntactic constituents. This is because there are likely to be many combinations of plausible words from the word lattice which form syntactically reasonable constituents but which may be ruled out on other grounds. For example, multiple possible placements of prepositional phrases frequently lead to ambiguous parsings (an effect frequently produced by text parsers) but either semantic knowledge or prosodic contours can be used to eliminate (or at least to rank order) erroneous and unlikely constructions. (For example, "John shot the bird in the tree" vs "John shot the bird in the wing" and "Give me the chair by the fireplace" vs "Give me the chair by tomorrow".) To allow immediate use of this information which syntax cannot

provide alone, SPARSER has an interface to the semantic component so that constituents can be verified directly without going through the control component. It will be a trivial modification to insert verification calls to pragmatics and prosodics when they become available. In the meantime, even semantic knowledge can be turned off; if the parser gets no information from the call to semantics, it proceeds without it.

Placement of a constituent in the WFST causes a number of things to happen. First, any monitors which have been set by the current theory at that position are activated. That is, for each configuration which was waiting for this constituent, a PUSH transition is made which uses the constituent as its input item. If no monitors have been set which can use this constituent, it is treated exactly as if it were the first word of an island: all the PUSH arcs which can use it are found in the grammar index and appropriate configurations and transitions (including lead-in transitions, if appropriate) are set up. Next, if there are any monitors for other theories which can use the constituent, notices are created and output to Control as was described above in the section on PUSH transitions.

Figure 4.6 shows SPARSER's map after our example island has been completely processed. The parsing results in the creation of a CAT N transition to configuration NP/N:30 using the word "analyses". The PUSH PP/ arc at state NP/N would cause configuration PP/:30 to be created. Similarly, PP/:22 would be created when the configuration NP/N:22 is picked up to be extended. The POP arc transitions from each of the configurations for state NP/N result in the formation of complete paths, resulting in the creation of two noun phrases ("chemical analyses" and "chemical"). Since there were no monitors for them, they result in the creation of configuration PP/PREP:14 and its subsequent paths.

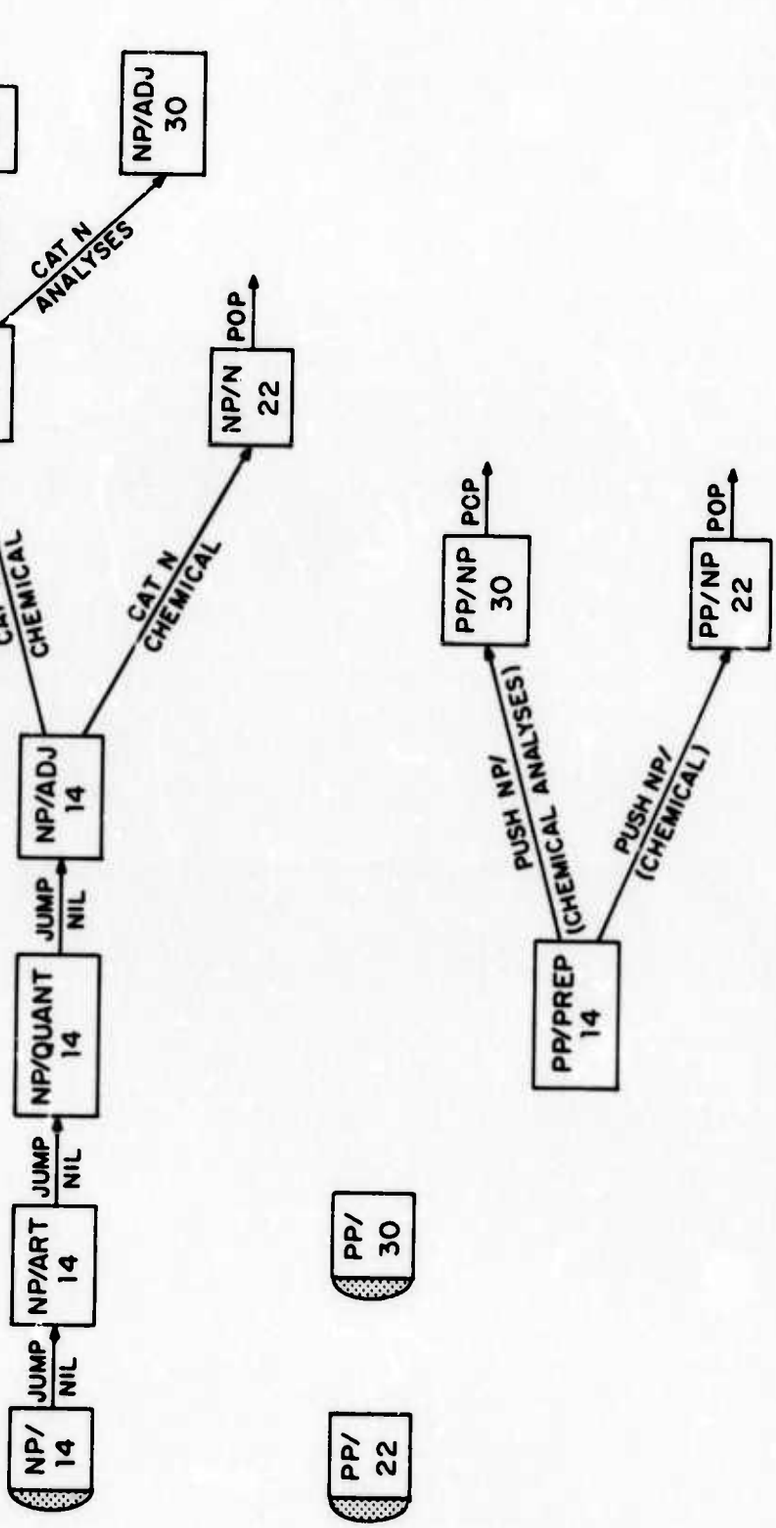

NP/ 14
JUMP NIL
NP/ART 14
JUMP NIL
NP/QUANT 14
JUMP NIL
NP/ADJ 14
CAT N CHEMICAL
CAT N CHEMICAL
NP/ADJ 22
NP/N 22
POP
CAT N ANALYSES
CAT N ANALYSES
NP/N 30
POP
NP/ADJ 30
PP/ 22
PP/ 30
PP/PREP 14
PUSH NP/ (CHEMICAL ANALYSES)
PUSH NP/ (CHEMICAL)
PP/NP 30
PCP
PP/NP 22
POP

Figure 4.6

4.3.4 Ending an Island

It may be the case that no path can be found from one end of an island to the other. (This would occur when all active configurations block.) In this case, there is no possible way that the island could form part of a grammatical string, so SPARSER can inform the control component that the theory is wrong.

When an active configuration is picked up to be extended and there is no word match at that point, the end of the island has been reached. That does not mean that no more transitions can be made, since arcs which do not test the input word can be taken as usual. Arcs which do use input cannot be taken, but they can be used to predict what sort of input would be acceptable at that position. For example, a CAT V arc which has a test requiring the verb to be untensed would allow SPARSER to predict an untensed verb beginning at the position of the current configuration. CAT and WRD arcs cause the prediction of syntactic categories and specific words, respectively, modified by the context-free test on the arc. TST arcs provide only the test which must be satisfied, and PUSH arcs cause a monitor to be set in the WFST as well as a TST monitor for the the look-ahead test (if any) on the arc.

More details on the nature of these predictions is given in the following section.

4.3.5 Ending a Theory

When all the islands of a theory have been processed in the manner just described, it is time to deal with the gaps between the islands. As we have seen, arcs in the grammar which can enter configurations at the left end of an island or which can leave configurations at the right end of an island can be used to make predictions about words that may be adjacent to the island. The prediction is a list of the arc, the configuration it would connect to, and an indication of whether the transition caused by the arc will enter the configuration from the left or leave it to the right.

If a gap between two islands is small enough that it may contain just one word, then it is likely that the arc which would process that word may have caused a prediction from both the left and right sides of the gap. If this is the case, and if the predictions intersect in a single possibility, it is highly probable that the word (or syntactic class) so predicted is correct. If the predictions do not intersect, parsing is continued from the active configurations which were not tried earlier because of their scores and from the configurations which

could begin constituents at the right end of an island. This continued parsing is an attempt to find a path which results in a common prediction across the gap. If that too fails, then the configurations which were saved because they could lead up a chain of PUSHes or POPs to new configurations are tried. If no possibilities are left to try and there is still no prediction to fill the gap, this information is noted, but it does not definitely mean that the islands are incompatible, since in some cases the gap could actually be filled by two words instead of one.

SPARSER has two kind of predictions - those which seem highly likely and those which seem less likely. A highly likely prediction, such as one which is made from both sides of a small gap, is output in the form of a proposal, which is a request to the rest of the system to find a word meeting the requirements of the proposal. A proposal contains:

1) the item being proposed, which is either a particular word or list of words (from a WRD arc), or a syntactic class (from a CAT arc), or NIL, meaning any word (from a TST arc)

2) the left and/or right boundary point(s) of the item

3) a test which the item must satisfy (the context free test from the arc)

4) the context of the proposal, i.e. the word match(es) on the left and/or right side of the item being proposed. (This is to help the lexical retrieval component take into account phonological phenomena which may occur across word boundaries.)

All predictions whether or not they are confident enough to become proposals are output as monitors. A monitor is a notification to the control component that if a word meeting the requirements of the monitor is somehow found (perhaps by the action of a proposal), it may be added to the theory. Thus a monitor acts like a demon which sits at a particular point in the word lattice and watches for the appearance of a word match which it can use. A monitor contains:

1) the item being monitored for (generally a syntactic category, but may be a word or a test)

2) the left or right boundary position of the item being monitored for

3) a test which the item must satisfy (same as for proposals)

4) the theory which generated the monitor

5) the arc in the grammar which will process the item if found

6) the configuration from which the prediction was made

7) a score, indicating roughly how important the monitor is, i.e. how much information is likely to be gained by processing an event for that monitor.
(Notice that monitors which are sent to the control component are very much like monitors which are set in the WFST by the occurrence of PUSH arcs.)

Once the proposals have been made and the monitors have been set, SPARSER bundles up the information it knows about the current theory, such as the configurations and transitions in the theory, any configurations which are still candidates for expansion, the constituents in the theory, the notices, proposals, and monitors which have been created, etc. and associates the bundle with the theory number. This insures that SPARSER will be able to pickup where it left off if it is later given the theory to process further.

4.3.6 Processing Multiple Theories

Thus far we have seen only the operations which SPARSER performs on a single theory, but we made the assumption in Section 4.1 that SPARSER would be given a number of theories to process in sequence. Let us now examine what will happen when the second (or nth) theory is processed.

SPARSER will no longer have a blank map and WFST; instead it will have all the configurations, transitions, and constituents which have been constructed by all previous theories. For concreteness, let us imagine that the theory (1 CHEMICAL 14 22) (2 ANALYSES 22 30) has been processed, resulting in the map shown in Figure 4.6. Now we are going to process a theory containing the island (4 NICKEL 16 22) (2 ANALYSES 22 30), which results in the map of Figure 4.7 where the configurations and transitions added by this theory are shown in dotted lines.

The process begins as usual with the creation of configuration NP/ADJ:16 and three possible lead-in transitions. The transitions for the two CAT N arcs, however terminate on configurations which already existed in the map, so the complete paths from configuration NP/:16 to configurations NP/N:30 and NP/N:22 will be discovered and processed, resulting in the construction of two new noun phrases. Those new constituents would then result in the creation of configuration PP/PREP:16 and two new transitions. Thus we have constructed only five new configurations and seven new transitions and have been able to take advantage of six old configurations and six old transitions.

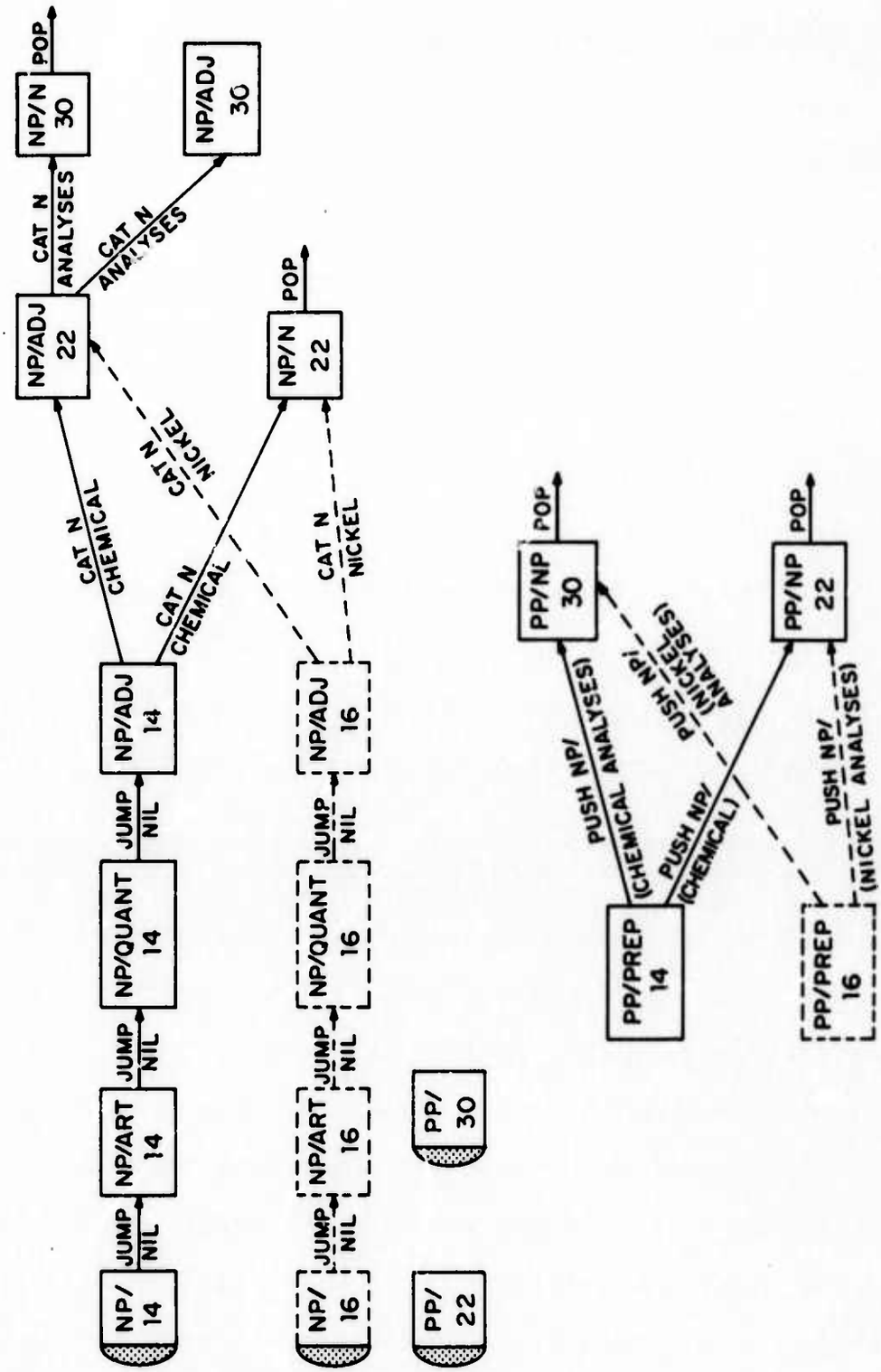

NP/N 30
POP
NP/ADJ 30
CAT N ANALYSES
CAT N ANALYSES
NP/ADJ 22
NP/N 22
POP
CAT N NICKEL
CAT N CHEMICAL
CAT N CHEMICAL
CAT N NICKEL
NP/ADJ 14
NP/ADJ 16
JUMP NIL
NP/QUANT 14
NP/QUANT 16
NP/ART 14
NP/ART 16
NP/ 14
NP/ 16
PP/ 30
PP/ 22
PP/NP 30
POP
PP/NP 22
POP
PUSH NP/ (CHEMICAL ANALYSES)
PUSH NP/ (NICKEL ANALYSES)
PUSH NP/ (CHEMICAL)
PUSH NP/ (NICKEL ANALYSES)
PP/PREP 14
PP/PREP 16

Figure 4.7

In this fashion any information which has once been discovered about a possible parse path is made available to any other path which can use it. No reparsing is ever done -- SPARSER merely realizes the existence of relevant configurations and transitions and incorporates them into the current theory.

If the new word (or words) in a theory are at the end (or in the middle) of an island, when SPARSER begins to parse the island it will discover the existing configurations and transitions from the previous theory. Whenever a transition which can be used in the current theory is discovered in the map, it and its terminating configuration are added to the syntactic part of the current theory. This is called tracing the transition. In addition, all paths beginning with that transition which do not require the next word of input are also included in the syntactic part of the theory. This is accomplished by tracing from the terminating configuration all transitions which use either the same word of input as the previous transition or no input word at all. (A similar process is used to trace backwards, i.e. right to left, when necessary.) When a configuration is reached which has no traceable transitions emanating from it, the tracing process stops. Since both transitions and configurations are stored in such a way as to facilitate

tracing (for example, each transition has a code attached to indicate whether or not it consumes or tests input), this process is considerably faster than creating that portion of the map in the first place. (To illustrate this, a theory was processed twice, once with an empty map and once starting with the map previously created; the time required for processing the theory fell from 47.5 seconds to 16.5.)

Configurations which can end traced paths are put on the active configurations list. If, when one of them is picked up for extension, it is discovered that the next word of input was used on a transition already in the map, the tracing process is repeated. If the next word of input is new (or at least has not caused any transitions from the configuration being considered) then parsing continues in the normal manner.

4.3.7 Processing Events

As was mentioned earlier, SPARSER can be called upon to add some new word matches to a theory it has previously processed. In this case, SPARSER is said to process an event. An event may be thought of rather abstractly as the discovery of a piece of information that has been syntactically proposed, monitored for, or noticed. Concretely, an event is a piece of data consisting of:

1) the old theory that proposed or set a monitor for the event

2) something to be added to the theory (a new word match or constituent)

4) the arc in the grammar which will process the new information

4) the configuration in the old theory which will be at one end of the transition created by the above arc.

When SPARSER is given an event, it retrieves from its tables the bundle of configurations, transition, etc. in the old theory. Then using the arc and the new word or constituent in the event, it creates the appropriate transition(s). Then processing continues as usual, that is, any complete paths are noticed and processed, and any new active configurations are extended, if possible.

New predictions may be made as a result of this increased information. (A record is kept of previous predictions so none are remade unless with a more liberal score.) Finally SPARSER returns the new, larger theory. This new theory may be processed as part of another event at some later time, thus gradually reducing the number and size of the gaps in the theory.

If an event results in filling the final gap in a theory, and if the resulting complete sequence of words can be parsed (as evidenced by the creation of a constituent which spans the utterance and which is parsed beginning in the initial state of the grammar, SPARSER notifies the control component of this fact, since the entire utterance may have been discovered. Of course, this may not be the correct solution -- it is up to the control component to look at the acoustic goodness, semantic meaningfulness, pragmatic likelihood, etc. of the result as well as the syntactic structure before declaring the utterance to have been understood. If for reasons other than syntactic, the utterance appears to be bad, the control component of the system could go on to try to find another, more suitable, possibility.

4.3.8 Summary

We have presented an overview of SPARSER together with several examples of its processing. The next chapter will delve more deeply into issues which have been ignored or glossed over so far, such as the relative scoring of configurations, predictions, and constituents and the various control strategies which may be used.

Chapter 5

More Details of the Parsing Process

5.1 INTRODUCTION

This chapter details the control strategy used by SPARSER to try to develop the most likely partial paths through an island. The problems of scoring the partial paths, monitors, proposals, and events are also discussed.

5.2 DEPTH vs. BREADTH

Chapter Two explored some of the arguments of top down vs. bottom up parsing methods, concluding that some mixture of the two would be best for processing speech. The parsing strategy used by SPARSER which was outlined in the previous chapter works bottom up when beginning to parse an island and when a constituent is created which was not monitored for by the current theory. It works top down after an island has been started and to make syntactic predictions at the ends of islands.

Both top down and bottom up techniques can be either depth or breadth first. Depth first processing takes at every step the first piece of information available (e.g. the first rule for the leftmost leaf of the parse tree so far constructed, or the first arc from the current state of an ATN grammar) and pursues its consequences. Breadth first processing considers at every step every possible next step of every alternative and pursues all paths in parallel.

Concretely, consider depth and breadth first processing using an ATN or MATN grammar. A top down, depth first parser (such as that which was described in Section 2.3.4) has only one state current at a time, and only one arc from that state is pursued at a time.

A top down, breadth first process for the same grammar would start with the initial state and try all the arcs emanating from it. Then, either by splitting into parallel processes or by keeping a list of subsequent states and processing them serially, all possible second steps would be made, then the process would be repeated until a parsing was found or all paths were exhausted. This would generally take much more space than the depth first process since many paths would have to be remembered at once instead of having just one stack which could be popped and reused when necessary.

The breadth first process might save some computation steps and might produce several ambiguous parsings simultaneously while the depth first process would find one before the others (the latter is a small difference, since both processes would have to be run to exhaustion to insure that all possible parsings had been found). In parsing speech, some mixture of breadth first and depth first processing can be extremely useful.

To illustrate an advantage of breadth first processing in the speech environment, consider what might happen if, during the processing of an island the parser picks up a configuration to extend which has several possible arcs emanating from it. If one arc is chosen and all the others are held as alternatives (i.e. depth first), but the chosen arc is wrong, all subsequent paths beginning with that arc would have to block before the alternatives would be tried. However, if the end of the island were reached before the success or failure of the first choice were confirmed, the only way that backup would ever take place would be to have one or more events add words to the theory so that the path could be extended until it failed. Since the gap would be likely to be filled by (incorrect) words predicted by the erroneous path, or by no words at all if the (incorrect) predictions were not satisfied, it is not at all clear how the process

would ever know to back up.

This problem cannot be eliminated completely without pursuing all alternatives to their fullest extent (a combinatorially unacceptable solution) but it can be modified to a great extent by a judicious combination of depth and breadth first processing to find the best path, not just the first one, through the island. This "best path" is not guaranteed to be the correct one, so it is possible to continue processing by extending paths with were suspended earlier.

SPARSER handles the problem by assigning a score to every configuration which reflects the likelihood of the path which terminates on that configuration to be correct. (What this likelihood means and the case where several paths may terminate on one configuration will be considered below.) The score can also be thought of as a measure of how good that configuration looks in relation to others as a candidate for extension. One question which was left unanswered in Chapter Four, how a subset of the active configurations is chosen for extension, can now be answered: the subset of maximally scoring configurations is chosen at each step until the maximal score of active configurations begins to fall. (The score on a configuration and the score of a path terminating on that configuration are the same thing -- we will use which

ever terminology seems most natural at the time. The reader should bear in mind that there is really no difference between the two.)

The result of this process is a sort of modified breadth first approach, where at one step all the alternatives are tried but at the next step only the best ones are chosen for further extension. This is similar to the best-first parser described by Paxton in [59] but it can be applied to the sort of partial paths which SPARSER generates rather than requiring the perfect information resulting from a strictly left to right approach. The success of this method is directly dependent on the relative accuracy of the scores which are assigned to the paths.

5.3 SCORING PATHS

Several attempts have been made to develop rigorous systems for parsing errorful or speech-like input based on probabilities [4, 5, 50, 76]. These attempts have all simplified the problem to such an extent that it is no longer realistic or extendible, e.g. by assuming the input is a sequence (rather than a lattice) of probability distributions, by assuming that all the necessary information is present in the search space to begin with

so the only problem is to find an optimal path through the space, by requiring a small vocabulary, and/or by limiting the grammar to be context free.

The ideal scoring mechanism for SPARSER would be one which accurately reflected at every step the probability that the path was correct. That is, given a path p_j which terminates on a state which has arcs a_1, ..., a_k emanating from it, one would want to choose the arc a_i which maximizes the probability that the resulting path is correct using Bayes' rule: $P(a_i|p_j) = P(a_i \cap p_i)/P(p_j)$. Taking the arc a_i would result in an extended path p_{j+1} whose probability would be used at the next step. In order to do this, it would be necessary to know, at any point in the parsing process, what the probability is that the next arc under consideration is correct, given that the entire path up to the current step is correct. In order to use this application of Bayes' rule it would be necessary to pre-calculate the probabilities for every possible path and partial path which could be generated -- a clearly impossible task since there are an infinite number of such paths.

As an approximation to this task, one might consider parsing a large number of randomly chosen sentences and keeping statistics on the parse paths taken in order to determine the frequency of the partial parse paths which

eventually lead to a correct parse. (The probability of paths not encountered during the statistics gathering could then be set to zero, or some appropriately small number.) Even this would be a formidable task, and would not really convey information about the best arc to take at a given point since the most probable arc could lead to a long path before blocking while it may cost relatively little to try a less probable arc first because it will lead to a block very quickly if it is incorrect.

Given that we cannot calculate the probabilities we need exactly, what is the next best option? If we ignore the effect of the path traversed up to the current point, but can say for any given state how likely each arc emanating from that state is to be correct, we would have a model which uses only local information rather than one which takes into account accurately all the left context which is available.

Since it was not practical to run large amounts of data through a parser in order to obtain accurate measurements even for the limited model, the author relied on considerable experience with ATN grammars to assign a weight to each arc of the grammar representing the intuitive likelihood that the arc (if it can be taken) is the correct one to choose from that state. These weights are small integers (0 through 5) -- the larger the weight

the more likely the arc.

The question might arise as to why the score of the word match used by an arc should not be used to influence the score of the path using it. SPARSER tries to treat each theory as independently as possible and to assign scores based only on the syntactic information which is available. The one exception to this rule is the semantic information which is used to score constituents. If lexical word match scores were used, the control component would not be able to separate the lexical goodness from the syntactic goodness of the theory and make judgments as to their relative importance. In a syntax-driven speech understanding system, however, it would probably be useful to combine lexical scores with syntactic information.

As was described in the previous chapter, when SPARSER begins to parse an island each possible partial path is begun by creating a configuration at the head of a transition for an arc which can use the current word. Rather arbitrarily, it was decided to give this configuration a score of one. This starts all partial paths out equally, a technique which is not quite accurate, since some contexts are more likely than others. For example, the words "to" and "for" are more likely to occur in prepositional phrases than in sentential complements. If this simplification appears to harm the

overall performance of SPARSER, it could be remedied by giving each state an a priori score similar to the weights on arcs. Configurations on lead-in paths are also given a score of one.

After the initial step, whenever a transition (other than a PUSH or POP) is made, the score of the subsequent configuration is set to be the sum of the score of the configuration being extended and the weight on the arc being used. If the scores were actual probabilities, they would be multiplied; since they are not, it was arbitrarily decided to add them.

When attempting to create a configuration which already exists (a situation encountered whenever two or more parse paths for the same theory merge), the configuration is given the maximum of the existing score and the score which would have been assigned had the configuration been created anew.

When a PUSH arc is encountered and a configuration created to begin the search for the required constituent, the score of that configuration is set to be the sum of the score of the configuration causing the PUSH, and the value (if any) of the look-ahead test on the PUSH arc. For example, upon encountering an arc such as (PUSH NP/ ((NPSTART) T T) ...) the look-ahead function

NPSTART returns a high integer value if the next word is a noun and a lower value if it is a verb (e.g. "accounting costs"), returns 0 if there is no next word, and returns an indication of failure if the next word cannot start a noun phrase. The score of the initial configuration for state NP/ will be slightly higher if the next word of input is a noun than if it is a verb. Of course, if the look-ahead function fails altogether, the configuration is not set up, although the monitor in the WFST remains. If there is no next word (i.e. at the end of an island) no extra term is added to compute the score of the new configuration.

When a constituent is completed (or found in the WFST) and a PUSH transition is about to be made, the score of the configuration on which the transition terminates is the sum of the score of the configuration being extended and half the sum of the weight on the arc and the score of the constituent itself. The score of the constituent is currently very ad hoc, being a function of the number of words in the constituent (less a function of the number of sub-constituents subsumed by this constituent, boosted if the constituent is a major one) and the score which is determined by semantic verification. (In the BBN system, the semantic component may assign several scores for various contexts in which the constituent may be used;

the maximum of these scores is used.) Thus semantically "good" constituents will boost the scores of the paths which use them more than semantically "bad" ones. Currently all constituent scores are positive; it would be interesting to experiment with negative scores as well in order to avoid incrementing even slightly scores of paths with unlikely (i.e. "bad") constituents.

Due to the level of effort required to gather accurate statistics on the relative frequencies of arcs, the current scores are admittedly ad hoc. It is not clear whether different scoring mechanisms would be better, however it is clear that the current scoring strategy is better than no scoring at all, as preliminary measurements indcate that the number of transitions created (as well as the number of configurations and predicions) is reduced about 25% by the current strategy.

(It is reasonable to ask why semantic scores are used to influence parse paths, since it was just argued that lexical scores should not be used in this way. Semantic scores may be more reliable than lexical ones because we are assuming that the utterance is semantically meaningful. Under this assumption, a constituent like "range remainder" as a noun-noun modifier analogous to "surplus money" should be ruled out as early as possible. Since such constituents cannot be ruled out on syntactic

grounds alone, since prosodic information (which might help to rule them out) is not available (see discussion in Section 7.2), and since they would seriously overrun the parser with a plethora of false paths if they were not rejected, it seems reasonable to permit semantics to influence the parser.)

Examples of parsing using this system of weighted configurations will be given in Chapter Six.

5.4 SCORING PREDICTIONS

The previous chapter discussed three ways in which SPARSER can make predictions about what could fill in gaps between islands. Monitors wait for the occurrence of a word in the word lattice (or a constituent in the WFST), proposals request a search for a particular set of words, and notices indicate the presence of a usable word in the word lattice (or a constituent in the WFST). Since the processing of a typical theory is likely to result in a number of predictions (see examples in Chapter Six), it is necessary to be able to order them so that predictions most likely to be correct or most likely to yield important information will be acted upon first. For example, it is more important to fill a gap between two islands than to extend a single island, since by filling

the gap one can check the consistency of information which was locally good in each island individually but may not be consistent when they are joined. Since two words can occur together in (usually) many contexts but longer sequences are generally more restrictive, adding a word to a one word island is likely to be less profitable in terms of the number of possible paths which are eliminated by the addition than adding a word to a multi-word island.

It is up to the syntactic component to indicate to the control component the relative importance attached to each notice and monitor; the higher the score, the stronger the prediction.

Several factors influence the score attached to predictions. One is the length of the island to which the prediction is attached. One word islands, if they are processed at all, yield very little information and many predictions, hence the predictions are not scored high. Proposals are less important if there is already a noticeable word in the word lattice (since that word is acoustically better than the word to be proposed, else it would have been found earlier. However, if a proposal fills a gap between two islands, it is given a higher score. Notices are boosted in importance if an entire constituent may be added and penalized if they will add onto a one word island. A table showing the current

scoring strategy for predictions follows:

PROPOSALS

WORDS no score needed

CLASSES 0 if only one gap remains to be filled

95 if something has already been noticed in the word lattice at this point

70 if the word will fill a gap

80 otherwise

NOTICES 30 if noticed from both sides of the gap

0 for a one word island

10 otherwise

40 if a constituent, not a word was noticed

MONITORS

WORDS 5 for a one word island

15 otherwise

CLASSES 0 for a one word island

10 otherwise

TESTS 10 in any case

These scores appear to work fairly well with the rest of the BBN SPEECHLIS system, but have been developed by a process of interaction with the other components (in order to make the scores of syntactic predictions commensurate with those of semantic predictions) and may be changed considerably as the entire system evolves.

Small syntactic classes (e.g. determiners and prepositions) are proposed in their entirety (that is, their elements are to be enumerated and given to the lexical matching component for verification) if the island which monitored for them is more than one word long. If a gap between two islands is small enough for just one word and if a syntactic class has been monitored for from both sides of the gap, it is proposed in its entirety also.

5.5 STRATEGY

Let us digress a bit to consider an important global issue. Why is it advantageous to process theories in the mannner described in Section 4.3? That is, why parse each island separately and attempt to predict from right to left as well as from left to right? It would seem that if two islands of words were separated by a gap large enough for several more words it might be reasonable to process left to right using a predictive method to find words to fill the gap. This would result, when the words found by prediction filled the gap, with a complete left context in which to parse the second island, thus constraining the processing of the second island instead of setting up partial parse paths for every possible context. Several arguments may be made against this seemingly plausible

approach.

First, consider what would happen if the first island in a theory were near the middle of the utterance. By the process outlined above, the parser would have to parse the entire word lattice (including predictions made along the way) up to the point of the first island. In the limit this would result in parsing the complete word lattice, so why bother with the island at all? Arguments against such a completely predictive parser were given in Section 2.2.1; basically it would result in serious combinatorial problems if one or more words at or near the beginning of the utterance were poor matches acoustically (a very likely happening).

Now consider the case where the first of two islands is wholly or partly wrong. The predictions made by the path through the first island might then result in more incorrect words being drawn into the theory and the error would not be discovered until the parser blocked completely (which it would not do if a consistent interpretation could be constructed from erroneous words). Even if the error were detected and the erroneous word replaced (a non-trivial task), the parsing from that word on (including the predictions) would have to be redone.

It may also be the case that the missing word immediately following the first island is a particularly bad match acoustically. It would take considerable backup for the poor but correct word to be proposed from the left, but if the gap were to be filled from right to left a stronger prediction could be made from both sides, resulting in acceptance of the correct word in that position before other better scoring but less syntactically consistent words at the same place.

Of course, pathological examples can be constructed to make either scheme look bad. Let us see if we can quantify the problem. Let us assume that at the end of an average island there are n possible predictions about the word which can follow, and that when any one of these predictions is satisfied it will generate n more predictions for the next word, and so on. Then in order to find a sequence of 3 words to fill the gap to the next island it would require $n+n^2+n^3$ steps (see Figure 5.1).

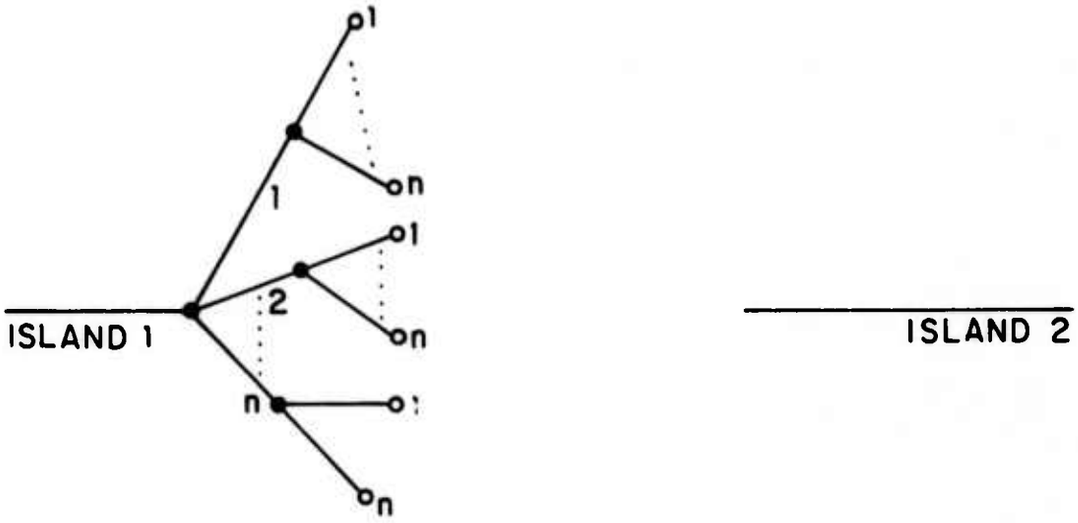


Figure 5.1

The Left to Right Approach

If the search proceeded from both sides of the gap, however, and a gap of one word could be filled in one step (by identical predictions from left and right since non-identical predictions would not be useful in completing the path), it would require only n+pm+1 steps (where there are m predictions on the average from the left end of an island and p partial paths begun for an average island; see Figure 5.2).

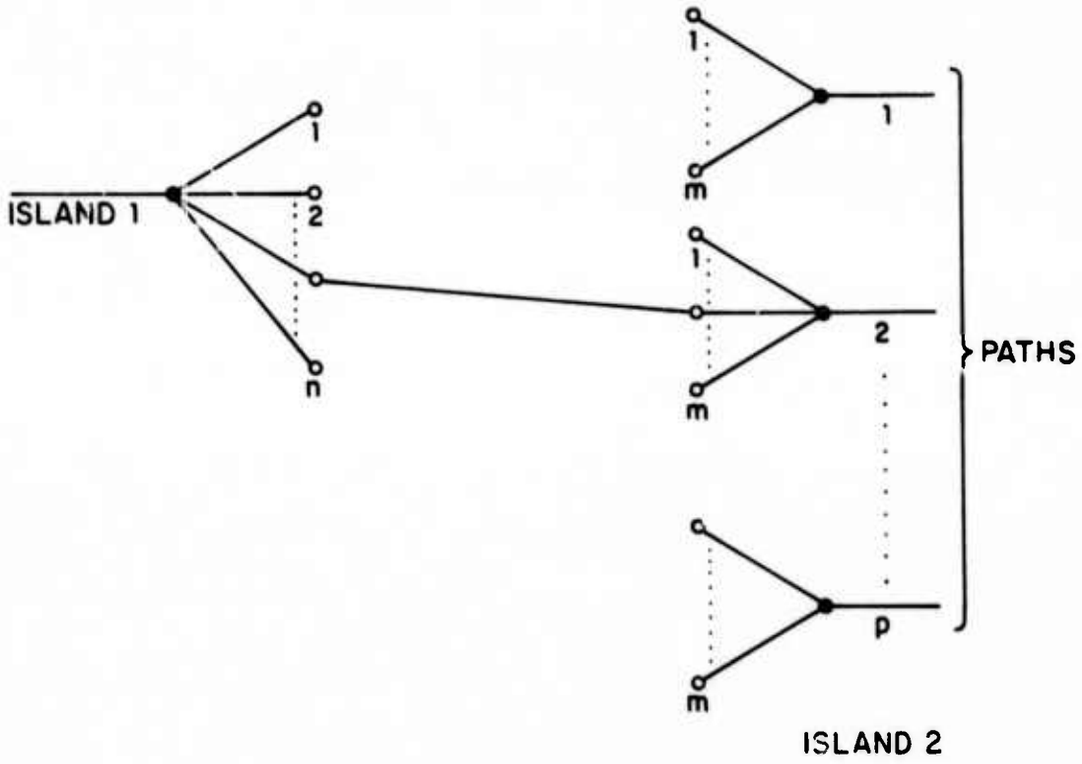


Figure 5.2

The Ends Toward Middle Approach

It is a good heuristic, though not a theorem (see Pohl [65]) that the best way to fill such a gap is to proceed from both ends toward the middle in breadth first fashion, not strictly alternating sides but choosing for extension whichever side has fewer nodes to be extended. Of course this technique assumes that all branches are equal and does not take into account that some branches may actually be much more reliable than others (by virtue of a good lexical score, for example), but it illustrates the advantage of being able to work from what ever reliable word are present in the utterance either forward or backward to fill in the gaps.

Chapter 6

Examples and Results

6.1 INTRODUCTION

SPARSER is written in INTERLISP [79] and runs on a PDP-10 under the TENEX operating system [11]. The program and initial data structures occupy approximately 90000 words of virtual memory. (The control component and the lexical retrieval and match component of the BBN speech understanding system occupy separate forks from the syntactic component.)

At the time of this writing, the algorithm controlling the decision-making process in the control component was undergoing revision and was not solidified into a function which could operate automatically. Rather, there were a number of primitive operations such as scanning an utterance (or some specified portion of it), creating theories, calling SPARSER with a theory or event, calling for the processing of proposals, etc., which could be invoked by a human simulator. The following examples were all produced in this mode, with the user acting as the control component in a way which could be modelled by later implementation.

Several conventions have been used in tracing the operation of SPARSER. Configurations are represented as NUMBER : STATE : POSITION (SCORE). For example, the configuration written as 30:NP/HEAD:23(39) is the configuration for state NP/HEAD at position 23 which has been given the (unique) number 30 and which currently has a score of 39. The creation of a transition is indicated by naming the type of arc causing the transition, the (unique) number of the transition, and the configurations at each end of the transition. For example,
CAT N TRANS #9 FROM 14:NP/DET:6(1) TO 15:NP/DET:19(4).

Annotations have been inserted within brackets { }; typeout in upper case was produced by the program.

6.2 EXAMPLE 1

This example parallels that given in Chapter Four. A word lattice was artificially created which contained only the following three word matches:

```
(1 SUMMER 12 16 100)
(2 WINTER 12 16 100)
(3 TRIP 16 21 100 -S)
```

(In this version of the system, regular inflectional endings are included in word matches after the element representing the score, hence the somewhat peculiar word match for the word "trips".) Two theories were

constructed, one for word matches 2 and 3, the other for 1 and 3. What follows is an annotated (but otherwise unedited except for considerations of spacing) transcript of SPARSER processing these two theories in sequence, using the MINIGRAMMAR of Figure 3.3 and Appendix I.

```
SPARSER PROCESSING THEORY 1:
0  12 WINTER 16 TRIP -S 21  30
          {This is a linear representation of the theory
          being processed.  The endpoints are 0 and 30,
          but the words occupy only the middle part of
          the utterance.}

STARTING AN ISLAND
"WINTER" TRYING CAT N ARC FROM NP/ADJ TO NP/ADJ
          {This is the first of two arcs retrieved from
          the index tables.}
  CAT N TRANS #1 FROM 1:NP/ADJ:12(1) TO 2:NP/ADJ:16(3)
          {The first transition is created, and since
          there is a CAT N arc which enters state NP/ADJ,
          a monitor is set up to monitor for nouns which
          end at position 12.}
   ENDING AT 12:
    MONITORING [  N  ]
  JUMP TRANS #2 FROM 3:NP/QUANT:12(1) TO 1:NP/ADJ:12(1)
          {Now the lead-in transitions are being created,
          along with the monitors for syntactic
          categories which may precede the newly
          constructed configurations.  Configurations
          along the lead-in path are all assigned a score
          of 1.}
   ENDING AT 12:
    MONITORING [  QUANT  ]
    MONITORING [  ADJ  ]
  JUMP TRANS #3 FROM 4:NP/ART:12(1) TO 3:NP/QUANT:12(1)
   ENDING AT 12:
    MONITORING [  ART  ]
  JUMP TRANS #4 FROM 5:NP/:12(1) TO 4:NP/ART:12(1)
          {The lead-in transitions are all made.  Now the
          second arc which can use the noun is about to
          be processed.}


"WINTER" TRYING CAT N ARC FROM NP/ADJ TO NP/N
```

```
  CAT N TRANS #5 FROM 1:NP/ADJ:12(1) TO 6:NP/N:16(6)
          {This is the second of the two arcs obtained
          from the index table for "winter". The lead-in
          transitions to configuration 1 have already
          been constructed, so they are not remade. Now
          we are ready to choose configurations to
          extend. The pool of candidates for extension
          contains configurations 2 and 6.}

SELECTED CONFIGS (6) FOR EXTENSION
          {Only this one is chosen because it has a
          higher score than configuration 2, since the
          use of a noun as a head noun of a noun phrase
          is more likely than its use as a modifier.}
PICKING UP CONFIG 6:NP/N:16(6) WITH WORD TRIP
 TRYING PUSH PP/ ARC
          {No action is taken about starting a
          configuration for state PP/ because the
          look-ahead test which checks that the next word
          can begin a prepositional phrase fails on the
          word trip.}
 TRYING POP ARC
  POP TRANS #6 FROM 6:NP/N:16(6)
          {Creating the POP transition completes a path
          from configuration 5. The path is expressed as
          a list of transition numbers. We are about to
          execute the path, that is, check the
          context-sensitive tests and do the register
          building actions along it.}
    EXECUTING PATH (4 3 2 5 6)
     BEGINNING AT TRANS 4, CONFIG 5
          {We must begin executing the path at the first
          transition, because no part of it has been
          executed before. Later we will see that it is
          possible to begin execution of a path in the
          middle, since the register contents are stored
          at each step.}
    DOING JUMP ARC FROM 5:NP/ TO 4:NP/ART
    DOING JUMP ARC FROM 4:NP/ART TO 3:NP/QUANT
    DOING JUMP ARC FROM 3:NP/QUANT TO 1:NP/ADJ
    DOING CAT ARC WITH WINTER FROM 1:NP/ADJ TO 6:NP/N
    DOING POP ARC FROM 6:NP/N
     TEST FAILED
          {The test failed because there is no
          determiner, and MINIGRAMMAR requires that
          singular, undetermined nouns can be complete
          noun phrases only if they are mass nouns.
          "Winter" is not marked as a mass noun in our
          dictionary, hence it will not parse as a
          complete noun phrase.}
```

```
SELECTED CONFIGS (2) FOR EXTENSION
          {Since extending configuration 6 did not do
          much for us, we go back to try the lower
          scoring configuration 2.}
PICKING UP CONFIG 2:NP/ADJ:16(3) WITH WORD TRIP
 TRYING CAT N ARC
  CAT N TRANS #7 FROM 2:NP/ADJ:16(3) TO 7:NP/N:21(8)
 TRYING CAT N ARC
  CAT N TRANS #8 FROM 2:NP/ADJ:16(3) TO 8:NP/ADJ:21(5)


SELECTED CONFIGS (7) FOR EXTENSION
          {Again, the higher scoring of the two active
          configurations, 7 and 8, is chosen.}
PICKING UP CONFIG 7:NP/N:21(8)
   STARTING AT 21:
    MONITORING [  PP/  ]
     SETTING UP CONFIG 9:PP/:21(8)
    MONITORING [ PREP ]
          {Since there is no next word to test, a
          configuration is set up to begin processing a
          prepositional phrase, and the syntactic
          categories which can begin such a phrase -- in
          this case, only one -- are monitored for.}
 TRYING POP ARC
  POP TRANS #9 FROM 7:NP/N:21(8)
    EXECUTING PATH (4 3 2 1 7 9)
     BEGINNING AT TRANS 1, CONFIG 1
          {Creation of the POP trans completed a path,
          the first part of which has already been
          executed. We can therefore pick up in the
          middle of the path and execute only the last
          three transitions.}
    DOING CAT ARC WITH WINTER FROM 1:NP/ADJ TO 2:NP/ADJ
    DOING CAT ARC WITH TRIP FROM 2:NP/ADJ TO 7:NP/N
    DOING POP ARC FROM 7:NP/N
****
    MADE #1 FROM 12 TO 21:
    NP ADJ NP N WINTER
               NU SG
       N TRIP
       NU PL
****
          {The path succeeds -- no determiner is needed
          since the head noun is plural -- and a
          constituent is constructed. The semantic
          component has been turned off for this example,
          so it adds nothing to the score which SPARSER
          assigns -- 5 points for each word in the
          constituent.}
SYN WEIGHT + SEM WT = 10 + 0 = 10
```

{No monitor exists in the WFST for a NP/ at this place, so the arcs (in MINIGRAMMAR there is only one) which could push for a NP are processed bottom up in exactly the same manner as the two arcs which could use a noun at the beginning of the island.}

```
NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #10 FROM 10:PP/PREP:12(1)
       TO 11:PP/NP:21(7)
   ENDING AT 12:
    MONITORING [  PREP  ]


SELECTED CONFIGS (11) FOR EXTENSION
PICKING UP CONFIG 11:PP/NP:21(7)
 TRYING POP ARC
  POP TRANS #11 FROM 11:PP/NP:21(7)


SELECTED CONFIGS (8) FOR EXTENSION
PICKING UP CONFIG 8:NP/ADJ:21(5)
   STARTING AT 21:
    MONITORING [  N  ]
    MONITORING [  N  ]
 ALL ARCS TRIED AT THIS CONFIG
```

{Now the theory has been processed. There follows a summary of the proposals, monitors, and notices constructed. The syntactic score assigned to the theory is given -- here just the score of the constituent constructed. Then there is a summary of statistics.}

```
PREDICTIONS:
MONITORING [  PREP  ] STARTING AT 21, SCORE 10
MONITORING [  N  ] STARTING AT 21, SCORE 10
MONITORING [  N  ] ENDING AT 12, SCORE 10
MONITORING [  QUANT  ] ENDING AT 12, SCORE 10
MONITORING [  ADJ  ] ENDING AT 12, SCORE 10
MONITORING [  ART  ] ENDING AT 12, SCORE 10
MONITORING [  PREP  ] ENDING AT 12, SCORE 10
PROPOSING (QUANT ART PREP) ENDING AT 12

FINISHED THEORY 1 WITH SYN SCORE 10

10 MONITORS SET
   0 WORDS
   7 CATEGORIES
   3 CONSTITS
   0 TESTS
```

```
3 PROPOSALS MADE
   0 WORDS
   3 CATEGORIES
0 NOTICES MADE
   0 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND  {in the WFST}
1 CONSTITS MADE
   0 FOR OTHER THEORIES
   1 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {Exclusive of tracing and fork interactions,
          this processing took 5.5 seconds.}


          {Now we are ready to process the second theory
          syntactically.}

SPARSER PROCESSING THEORY 2:
0  12 SUMMER 16 TRIP -S 21  30

STARTING AN ISLAND
"SUMMER" TRYING CAT N ARC FROM NP/ADJ TO NP/ADJ
  CAT N TRANS #12 FROM 1:NP/ADJ:12(1) TO 2:NP/ADJ:16(3)
          {This transition completes a path which
          includes transitions and configurations
          constructed during the previous theory.}
    EXECUTING PATH (4 3 2 12 7 9)
     BEGINNING AT TRANS 12, CONFIG 1
    DOING CAT ARC WITH SUMMER FROM 1:NP/ADJ TO 2:NP/ADJ
    DOING CAT ARC WITH TRIP FROM 2:NP/ADJ TO 7:NP/N
    DOING POP ARC FROM 7:NP/N
****
    MADE #2 FROM 12 TO 21:
    NP ADJ NP N SUMMER
             NU SG
       N TRIP
       NU PL
****
SYN WEIGHT + SEM WT = 10 + 0 = 10


NP/ WAS PUSHED FOR AT CONFIG 10
          {This time there are monitors in the WFST, one
          which is looking for a NP starting at position
          12 and one which is looking for a NP ending at
          position 21.  One transition is sufficient to
          satisfy both of these, and the preposition
          needed to complete a PP/ is monitored for.}
 PUSH NP/ TRANS #13 FROM 10:PP/PREP:12(1)
```

```
       TO 11:PP/NP:21(8)
NP/ MAY LEAD TO CONFIG 11
```

{This is caused by the fact that there was a monitor for a noun phrase ending at configuration 11 -- the one created when constituent 1 was made. The transition which would be set up is the transition just created, so it is not remade.

All of the processing which resulted from the completion of a constituent is finished; however there are monitors still to be set for configurations along the path.}

```
   ENDING AT 12:
    MONITORING [  PREP  ]
   ENDING AT 12:
    MONITORING [  N  ]
   ENDING AT 12:
    MONITORING [  QUANT  ]
    MONITORING [  ADJ  ]
   ENDING AT 12:
    MONITORING [  ART  ]
```

{Since each monitor consists of the item being monitored for, its associated test (if any), the theory which is to be notified when the monitor is satisfied, and the configuration and arc causing the monitor, monitors must be made anew each time one of the elements changes, although some of the list structure can be shared, hence the seeming proliferation of monitors.}

{Now SPARSER processes the other arc which could use the word "summer".}

```
"SUMMER" TRYING CAT N ARC FROM NP/ADJ TO NP/N
  CAT N TRANS #14 FROM 1:NP/ADJ:12(1) TO 6:NP/N:16(6)
    EXECUTING PATH (4 3 2 14 6)
     BEGINNING AT TRANS 14, CONFIG 1
    DOING CAT ARC WITH SUMMER FROM 1:NP/ADJ TO 6:NP/N
    DOING POP ARC FROM 6:NP/N
     TEST FAILED
```

{Because "summer" cannot be a complete noun phrase in this grammar.}

```
SELECTED CONFIGS (11) FOR EXTENSION
PICKING UP CONFIG 11:PP/NP:21(8)
  TRACING POP TRANS 11 FROM 11:PP/NP:21(8)
```

{This transition was created before, but is now made part of the current theory. It does not complete a path or cause any further action. If it had a terminating configuration, i.e. if

a transition other than a POP transition had been traced, the terminating configuration would have been placed on the list of possible configurations to extend.}

```
SELECTED CONFIGS (6) FOR EXTENSION
PICKING UP CONFIG 6:NP/N:16(6) WITH WORD TRIP
  TRACING POP TRANS 6 FROM 6:NP/N:16(6)


SELECTED CONFIGS (2) FOR EXTENSION
PICKING UP CONFIG 2:NP/ADJ:16(3) WITH WORD TRIP
  TRACING CAT N TRANS 8 USING "TRIP" FROM
       2:NP/ADJ:16(3) TO 8:NP/ADJ:21(5)
   STARTING AT 21:
    MONITORING [  N  ]
    MONITORING [  N  ]
          {There are two noun arcs leaving state  NP/ADJ,
          hence two monitors.}


SELECTED CONFIGS (8) FOR EXTENSION
PICKING UP CONFIG 8:NP/ADJ:21(5)
 ALL ARCS TRIED AT THIS CONFIG



PREDICTIONS:
MONITORING [  N  ] STARTING AT 21, SCORE 10
MONITORING [  PREP  ] ENDING AT 12, SCORE 10
MONITORING [  N  ] ENDING AT 12, SCORE 10
MONITORING [  QUANT  ] ENDING AT 12, SCORE 10
MONITORING [  ADJ  ] ENDING AT 12, SCORE 10
MONITORING [  ART  ] ENDING AT 12, SCORE 10
PROPOSING (PREP QUANT ART) ENDING AT 12

FINISHED THEORY 2 WITH SYN SCORE 10

6 MONITORS SET
   0 WORDS
   6 CATEGORIES
   0 CONSTITS
   0 TESTS
3 PROPOSALS MADE
   0 WORDS
   3 CATEGORIES
```

```
0 NOTICES MADE
   0 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
1 CONSTITS MADE
   0 FOR OTHER THEORIES
   1 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {The processing of this theory took
          approximately 4.5 seconds.}
```

This example has shown the trace produced by running SPARSER on input which is analogous to the example presented with illustrations of the map in Chapter Four. The interested reader is urged to draw his own maps while reading the following examples in order to best understand the dynamic operation of SPARSER.

6.3 EXAMPLE 2

This example also uses MINIGRAMMAR, but shows how SPARSER helps to put together an entire utterance by noticing words in the word lattice and creating new theories by processing events. A word lattice was artificially created which contained all and only the words in the utterance "The final trip of the year". Since "trip" and "year" have a fairly strong semantic association, a theory was created for those two words in order to begin processing.

In order to prevent the reader from being overwhelmed with the extensive output of the trace, less relevant portions of the output in this and subsequent examples have been deleted to produce a more abbreviated protocol.

```
SPARSER PROCESSING THEORY 1:
0  6 TRIP 10   14 YEAR 16


STARTING AN ISLAND
          {Each island in this theory is processed
          exactly the way "winter" was processed at the
          beginning of Example 1.}
"TRIP" TRYING CAT N ARC FROM NP/ADJ TO NP/ADJ
  CAT N TRANS #1 FROM 1:NP/ADJ:6(1) TO 2:NP/ADJ:10(3)
   ENDING AT 6:
    MONITORING [  N  ]
  JUMP TRANS #2 FROM 3:NP/QUANT:6(1) TO 1:NP/ADJ:6(1)
   ENDING AT 6:
    MONITORING [  QUANT  ]
    MONITORING [  ADJ  ]
     NOTICING "FINAL"
          {The word "final" is in the word lattice, so a
          notice is created for it.}
  JUMP TRANS #3 FROM 4:NP/ART:6(1) TO 3:NP/QUANT:6(1)
   ENDING AT 6:
    MONITORING [  ART  ]
  JUMP TRANS #4 FROM 5:NP/:6(1) TO 4:NP/ART:6(1)
          {Now that the first transition has been started
           up, the second is processed.}


"TRIP" TRYING CAT N ARC FROM NP/ADJ TO NP/N
  CAT N TRANS #5 FROM 1:NP/ADJ:6(1) TO 6:NP/N:10(6)


          {Now the higher scoring of the two active
          configurations is chosen for extension.}
SELECTED CONFIGS (6) FOR EXTENSION
PICKING UP CONFIG 6:NP/N:10(6)
   STARTING AT 10:
    MONITORING [  PP/  ]
     SETTING UP CONFIG 7:PP/:10(6)
```

```
    MONITORING [ PREP ]
     NOTICING "OF"
 TRYING POP ARC
  POP TRANS #6 FROM 6:NP/N:10(6)
    EXECUTING PATH (4 3 2 5 6)
     BEGINNING AT TRANS 4, CONFIG 5
    DOING JUMP ARC FROM 5:NP/ TO 4:NP/ART
    DOING JUMP ARC FROM 4:NP/ART TO 3:NP/QUANT
    DOING JUMP ARC FROM 3:NP/QUANT TO 1:NP/ADJ
    DOING CAT ARC WITH TRIP FROM 1:NP/ADJ TO 6:NP/N
    DOING POP ARC FROM 6:NP/N
     TEST FAILED
          {Because "trip" cannot be a complete noun
          phrase.}


STARTING AN ISLAND
"YEAR" TRYING CAT N ARC FROM NP/ADJ TO NP/ADJ
  CAT N TRANS #7 FROM 8:NP/ADJ:14(1) TO 9:NP/ADJ:16(3)
   ENDING AT 14:
    MONITORING [  N  ]
  JUMP TRANS #8 FROM 10:NP/QUANT:14(1) TO 8:NP/ADJ:14(1)
   ENDING AT 14:
    MONITORING [  QUANT  ]
    MONITORING [  ADJ  ]
  JUMP TRANS #9 FROM 11:NP/ART:14(1) TO 10:NP/QUANT:14(1)
   ENDING AT 14:
    MONITORING [  ART  ]
     NOTICING "THE"
  JUMP TRANS #10 FROM 12:NP/:14(1) TO 11:NP/ART:14(1)


"YEAR" TRYING CAT N ARC FROM NP/ADJ TO NP/N
  CAT N TRANS #11 FROM 8:NP/ADJ:14(1) TO 13:NP/N:16(6)


SELECTED CONFIGS (13) FOR EXTENSION
PICKING UP CONFIG 13:NP/N:16(6)
 TRYING POP ARC
  POP TRANS #12 FROM 13:NP/N:16(6)
    EXECUTING PATH (10 9 8 11 12)
     BEGINNING AT TRANS 10, CONFIG 12
    DOING JUMP ARC FROM 12:NP/ TO 11:NP/ART
    DOING JUMP ARC FROM 11:NP/ART TO 10:NP/QUANT
    DOING JUMP ARC FROM 10:NP/QUANT TO 8:NP/ADJ
    DOING CAT ARC WITH YEAR FROM 8:NP/ADJ TO 13:NP/N
    DOING POP ARC FROM 13:NP/N
     TEST FAILED
```

```
PREDICTIONS:
NOTICING (2 THE 12 14 100 0), SCORE 0
NOTICING (3 OF 10 12 100 0), SCORE 0
NOTICING (5 FINAL 2 6 100 0), SCORE -5
MONITORING [  N  ] ENDING AT 14, SCORE 0
MONITORING [  QUANT  ] ENDING AT 14, SCORE 0
MONITORING [  ADJ  ] ENDING AT 14, SCORE 0
MONITORING [  ART  ] ENDING AT 14, SCORE 0
MONITORING [  PREP  ] STARTING AT 10, SCORE 0
MONITORING [  N  ] ENDING AT 6, SCORE 0
MONITORING [  QUANT  ] ENDING AT 6, SCORE 0
MONITORING [  ADJ  ] ENDING AT 6, SCORE 0
MONITORING [  ART  ] ENDING AT 6, SCORE 0
          {No syntactic class predictions are made,
          because words in the relevant classes were
          already in the word lattice and were noticed.}

FINISHED THEORY 1 WITH SYN SCORE 0
          {The score of 0 does not mean that this is not
          a good theory, but that no constituents have
          been constructed thus far.}

10 MONITORS SET
   0 WORDS
   9 CATEGORIES
   1 CONSTITS
   0 TESTS
0 PROPOSALS MADE
   0 WORDS
   0 CATEGORIES
3 NOTICES MADE
   3 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
0 CONSTITS MADE
   0 FOR OTHER THEORIES
   0 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {This processing took 5.9 seconds.}
```

{The three words in the word lattice which were adjacent to the words in the theory were all noticed. The score for the notice for "final" was lower in weight because adjectives are content words which are frequently combined with other words by the semantic component. If content words are not combined by Semantics, it is less likely that the combination is correct on strictly syntactic grounds. Even though

Semantics was not operating in this example, SPARSER gives precedence to notices which could not have been constructed except by the use of syntactic knowledge.

The event notices for the words "of" and "the" were given identical scores, but it chanced that the event for "of" was first on the list of events in the control component, so we will process that event first.}

```
SYNTAX PROCESSING EVENT FOR THEORY#1
  WITH NEW WORD (10 OF 12)
  TO GET NEW THEORY#2:
0  6 TRIP 10 OF 12   14 YEAR 16


SELECTED CONFIGS (7) FOR EXTENSION
PICKING UP CONFIG 7:PP/:10(6) WITH WORD OF
 TRYING CAT PREP ARC
  CAT PREP TRANS #13 FROM 7:PP/:10(6) TO 14:PP/PREP:12(11)


SELECTED CONFIGS (14) FOR EXTENSION
PICKING UP CONFIG 14:PP/PREP:12(11)
   STARTING AT 12:
    MONITORING [  NP/  ]
     SETTING UP CONFIG 15:NP/:12(11)
    MONITORING [ N ]
    MONITORING [ ADJ ]
    MONITORING [ ART ]
     NOTICING  THE"
    MONITORING [ QUANT ]
 ALL ARCS TRIED AT THIS CONFIG
```

{No more work need be done for this event. The predictions made for the previous theory which are still relevant are made for this theory also.}

```
PREDICTIONS:
NOTICING (2 THE 12 14 100 0), SCORE 10
NOTICING (2 THE 12 14 100 0), SCORE 0
```

{This word match was noticed from both sides of the gap. The higher score reflects the fact that it extends a two word island.}

```
NOTICING (5 FINAL 2 6 100 0), SCORE 5
```

```
MONITORING [  N  ] STARTING AT 12, SCORE 10
MONITORING [  ADJ  ] STARTING AT 12, SCORE 10
MONITORING [  ART  ] STARTING AT 12, SCORE 10
MONITORING [  QUANT  ] STARTING AT 12, SCORE 10
MONITORING [  N  ] ENDING AT 14, SCORE 0
MONITORING [  QUANT  ] ENDING AT 14, SCORE 0
MONITORING [  ADJ  ] ENDING AT 14, SCORE 0
MONITORING [  ART  ] ENDING AT 14, SCORE 0
MONITORING [  N  ] ENDING AT 6, SCORE 10
MONITORING [  QUANT  ] ENDING AT 6, SCORE 10
MONITORING [  ADJ  ] ENDING AT 6, SCORE 10
MONITORING [  ART  ] ENDING AT 6, SCORE 10
PROPOSING (QUANT ART) ENDING AT 6
```

{Since this event did not lead to a grammatically impossible sequence of words, SPARSER notifies the control component that it is all right to create a new theory which incorporates the new word.}

```
CREATING THEORY 2:
0  6 TRIP 10 OF 12   14 YEAR 16
 WITH SYN SCORE 0


13 MONITORS SET
   0 WORDS
   12 CATEGORIES
   1 CONSTITS
   0 TESTS
2 PROPOSALS MADE
   0 WORDS
   2 CATEGORIES
3 NOTICES MADE
   3 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
0 CONSTITS MADE
   0 FOR OTHER THEORIES
   0 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
```

{It took only 2.15 seconds to process this event.}

{Now the best event in the control component's queue is the one which adds the word "the" to the theory just created. The syntactic processing here is especially interesting.}

```
SYNTAX PROCESSING EVENT FOR THEORY#2
  WITH NEW WORD (12 THE 14)
  TO GET NEW THEORY#3:
0  6 TRIP 10 OF 12 THE 14 YEAR 16


SELECTED CONFIGS (15) FOR EXTENSION
PICKING UP CONFIG 15:NP/:12(11) WITH WORD THE
 TRYING CAT ART ARC
  CAT ART TRANS #14 FROM 15:NP/:12(11) TO 11:NP/ART:14(16)
    EXECUTING PATH (14 9 8 11 12)
     BEGINNING AT TRANS 14, CONFIG 15
```

{This path executes successfully, and a constituent is built.}

```
****
    MADE #1 FROM 12 TO 16:
    NP ART THE
       N YEAR
       NU SG
****
```

NOTIFYING THEORY#2 ABOUT CONSTITUENT #1

{Since a monitor was placed in the WFST during the last call to SPARSER, a notice is created. In this case, the control component will recognize that this event adds no new word matches to the theory being created now, so the event will never be processed. It might have been the case, however, that the constituent just constructed included word matches new to this theory; in that case the notice would be useful.}

```
SYN WEIGHT + SEM WT = 10 + 0 = 10

NP/ WAS PUSHED FOR AT CONFIG 14
  PUSH NP/ TRANS #15 FROM 14:PP/PREP:12(11)
       TO 16:PP/NP:16(18)
 TRYING JUMP NP/ART ARC
  JUMP TRANS #16 FROM 15:NP/:12(11) TO 17:NP/ART:12(15)
```

{This JUMP transition was made as a result of continuing to try the arcs from configuration 15. The creation of transition #14 completed a path and caused all the bottomup processing on the constituent, but after that was finished other arcs which were still pending, e.g. the JUMP arc, were tried.}

```
SELECTED CONFIGS (16) FOR EXTENSION
PICKING UP CONFIG 16:PP/NP:16(18)
 TRYING POP ARC
  POP TRANS #17 FROM 16:PP/NP:16(18)
    EXECUTING PATH (13 15 17)
     BEGINNING AT TRANS 13, CONFIG 7
****
    MADE #2 FROM 10 TO 16:
    PP PREP OF
       NP ART THE
          N YEAR
          NU SG
****
          NOTIFYING THEORY#1 ABOUT CONSTITUENT #2
          NOTIFYING THEORY#2 ABOUT CONSTITUENT #2
           {Same comment as above applies here.}
SYN WEIGHT + SEM WT = 15 + 0 = 15


PP/ WAS PUSHED FOR AT CONFIG 6
  PUSH PP/ TRANS #18 FROM 6:NP/N:10(6) TO 13:NP/N:16(15)


SELECTED CONFIGS (13) FOR EXTENSION
PICKING UP CONFIG 13.NP/N:16(15)
          {There are no new arcs to try from this
          configuration.}


"THE" TRYING CAT ART ARC FROM STATE NP/ TO CONFIG 11
          {This is the arc which caused "the" to be
          noticed to the left of the word "year". Since
          the relevant transition has already been made,
          no further processing is necessary.}


PREDICTIONS:
NOTICING (5 FINAL 2 6 100 0), SCORE 5
MONITORING [  N  ] ENDING AT 6, SCORE 10
MONITORING [  QUANT  ] ENDING AT 6, SCORE 10
MONITORING [  ADJ  ] ENDING AT 6, SCORE 10
MONITORING [  ART  ] ENDING AT 6, SCORE 10
PROPOSING (QUANT ART) ENDING AT 6


CREATING THEORY 3:
0  6 TRIP 10 OF 12 THE 14 YEAR 16
 WITH SYN SCORE 15
```

```
4 MONITORS SET
   0 WORDS
   4 CATEGORIES
   0 CONSTITS
   0 TESTS
0 PROPOSALS MADE
   0 WORDS
   0 CATEGORIES
4 NOTICES MADE
   1 WORDS
   0 CONSTITS(THIS THEORY)
   3 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
2 CONSTITS MADE
   0 FOR OTHER THEORIES
   2 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {This event took 6.4 seconds.}


          {Now the best event is that which adds the word
          "final".}


SYNTAX PROCESSING EVENT FOR THEORY#3
  WITH NEW WORD (2 FINAL 6)
  TO GET NEW THEORY#4:
0  2 FINAL 6 TRIP 10 OF 12 THE 14 YEAR 16


"FINAL" TRYING (CAT ADJ--) FROM STATE NP/QUANT TO CONFIG 3
 CAT ADJ TRANS #19 FROM 18:NP/QUANT:2(4)
       TO 3:NP/QUANT:6(8)
   ENDING AT 2:
    MONITORING [  QUANT  ]
    MONITORING [  ADJ  ]
  JUMP TRANS #20 FROM 19:NP/ART:2(1) TO 18:NP/QUANT:2(4)
   ENDING AT 2:
    MONITORING [  ART  ]
     NOTICING "THE"
  JUMP TRANS #21 FROM 20:NP/:2(1) TO 19:NP/ART:2(1)


PREDICTIONS:
NOTICING (6 THE 0 2 100 0), SCORE 10
MONITORING [  QUANT  ] ENDING AT 2, SCORE 10
MONITORING [  ADJ  ] ENDING AT 2, SCORE 10
MONITORING [  ART  ] ENDING AT 2, SCORE 10
PROPOSING (QUANT ADJ ART) STARTING AT 6
PROPOSING (QUANT ART) ENDING AT 2
```

```
CREATING THEORY 4:
0  2 FINAL 6 TRIP 10 OF 12 THE 14 YEAR 16
 WITH SYN SCORE 15


3 MONITORS SET
   0 WORDS
   3 CATEGORIES
   0 CONSTITS
   0 TESTS
2 PROPOSALS MADE
   0 WORDS
   2 CATEGORIES
1 NOTICES MADE
   1 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
0 CONSTITS MADE
   0 FOR OTHER THEORIES
   2 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {This took 4.25 seconds.}


          {Now the best event is that which adds the word
          "the" to the beginning of the theory just
          created. This completes the utterance, and
          SPARSER finds a successful parse of the entire
          sentence.}

SYNTAX PROCESSING EVENT FOR THEORY#4
  WITH NEW WORD (0 THE 2)
  TO GET NEW THEORY#5:
0 THE 2 FINAL 6 TRIP 10 OF 12 THE 14 YEAR 16


"THE" TRYING (CAT ART --) FROM STATE NP/ TO CONFIG 19
  CAT ART TRANS #22 FROM 21:NP/:0(5) TO 19:NP/ART:2(10)
    EXECUTING PATH (22 20 19 2 5 6)
     BEGINNING AT TRANS 22, CONFIG 21
****
    MADE #3 FROM 0 TO 10:
    NP ART THE
       ADJ FINAL
       N TRIP
       NU SG
****
SYN WEIGHT + SEM WT = 15 + 0 = 15
```

```
NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #23 FROM 22:PP/PREP:0(1) TO
 23:PP/NP:10(10)

    EXECUTING PATH (22 20 19 2 5 18 12)
     BEGINNING AT TRANS 18, CONFIG 6
****
    MADE #4 FROM 0 TO 16:
    NP ART THE
       ADJ FINAL
       N TRIP
       NU SG
       PP PREP OF
          NP ART THE
             N YEAR
             NU SG
****
SYN WEIGHT + SEM WT = 30 + 0 = 30

NP/ WAS PUSHED FOR AT CONFIG 22
  PUSH NP/ TRANS #24 FROM 22:PP/PREP:0(1) TO
 16:PP/NP:16(18)


CREATING THEORY 5:
0 THE 2 FINAL 6 TRIP 10 OF 12 THE 14 YEAR 16
 WITH SYN SCORE 35
```

This example has shown the way SPARSER notices words in the word lattice and helps to guide the construction of a theory which spand the entire utterance. In this case, the control strategy used was completely syntax oriented, that is, no semantic events were constructed or processed. Of course, word lattices generally will not contain all and only the correct words of the utterance. The next example demonstrates how SPARSER can help to select the best set of words from a more complex word lattice.

6.4 EXAMPLE 3

This example is more realistic than the previous two -- it shows the operation of SPARSER in the context of an utterance which has been automatically segmented and labeled, with the lexical retrieval and match component in operation. This example also uses the SPEECHGRAMMAR of Apendix II.

The utterance "What is the registration fee?" was spoken by an adult male speaker in a quite room and was recorded on tape. The utterance was then dThe utterance was automatically digitized and passed through the segmentation and labeling routines of the BBN speech understanding system. The initial scan of the utterance, using the lexical retrieval compnent, produced a word lattice of fifteen entries, including several for inflectional endings. (In this version of the system, they were not combined with the root form into a single word match, and hence could match even without a root word.) The format for a word match is:
(NUMBER WORD LEFT-END RIGHT-END LEXICAL-SCORE).

```
(2 WHAT 0 3 191)
(3 ONE 0 3 189)
(11 WHEN 0 3 102)
(9 THE 4 6)
(1 REGISTRATION 6 19 237)
(10 REGISTRATION 7 19 103)
(5 HAS 9 12 121)
(8 THIS 9 12 115)
(15 THE 9 11 90)
(6 -EST 10 13 118)
(12 -EST 10 14 101)
(13 IS 10 12 97)
(14 -ES 10 12 97)
(7 TRIP 12 17 116)
(4 FEE 19 23 155)
```

The two best matches, for "what" and "registration", appear to be good candidates for a theory, so we begin by building and processing that theory.

```
SPARSER PROCESSING THEORY 1:
0 WHAT 3   6 REGISTRATION 19  23

STARTING AN ISLAND
STARTING AT LEFT END OF SENTENCE
          {Knowing that it is not necessary to go through
          the usual startup procedure for islands when
          beginning an island at position 0, SPARSER
          starts with a configuration for state S/ at
          position 0.}
SELECTED CONFIGS (1) FOR EXTENSION
PICKING UP CONFIG 1:S/:0(1) WITH WORD WHAT
 TRYING JUMP S/Q ARC
  JUMP TRANS #1 FROM 1:S/:0(1) TO 2:S/Q:0(6)

SELECTED CONFIGS (2) FOR EXTENSION
PICKING UP CONFIG 2:S/Q:0(6) WITH WORD WHAT
```

```
 TRYING PUSH NP/ ARC
      MONITORING  [ NP/ ]
         SETTING UP CONFIG 3:NP/:0(11)
 TRYING CAT QWORD ARC
  CAT QWORD TRANS #2 FROM 2:S/Q:0(6) TO 4:S/NP:3(11)


SELECTED CONFIGS (3 4) FOR EXTENSION
           {This time two active configurations have the
           same maximal score, so they are both
           processed.}
PICKING UP CONFIG 3:NP/:0(11) WITH WORD WHAT
 TRYING CAT QDET ARC
  CAT QDET TRANS #3 FROM 3:NP/:0(11) TO 5:NP/ORD:3(16)

PICKING UP CONFIG 4:S/NP:3(11)
   STARTING AT 3:
    MONITORING [  MODAL  ]
    MONITORING [  V  ]
 TRYING POP ARC
  POP TRANS #4 FROM 4:S/NP:3(11)
    EXECUTING PATH (1 2 4)
     BEGINNING AT TRANS 1, CONFIG 1
    DOING JUMP ARC WITH WHAT FROM 1:S/ TO 2:S/Q
    DOING CAT ARC WITH WHAT FROM 2:S/Q TO 4:S/NP
    DOING POP ARC FROM 4:S/NP
     TEST FAILED
           {This test failed because the grammar does not
           allow "what" to be a complete sentence.}


SELECTED CONFIGS (5) FOR EXTENSION
PICKING UP CONFIG 5:NP/ORD:3(16)
   STARTING AT 3:
    MONITORING [  QUANT/  ]
     SETTING UP CONFIG 6:QUANT/:3(16)
           {Here all the words which can start
           quantifiers, like "a hundred" or "point five",
           are proposed. The grammar does not preclude a
           quantifier following a question-determiner,
           e.g. "What three men traveled to Spain?".}
    MONITORING [ INTEGER ]
    MONITORING [ ZERO ]
     PROPOSING "ZERO"
    MONITORING [ NO ]
     PROPOSING "NO"
    MONITORING [ POINT ]
     PROPOSING "POINT"
    MONITORING [ A ]
     PROPOSING "A"
```

```
 TRYING JUMP NP/QUANT ARC
  JUMP TRANS #5 FROM 5:NP/ORD:3(16) TO 7:NP/QUANT:3(21)


SELECTED CONFIGS (7) FOR EXTENSION
PICKING UP CONFIG 7:NP/QUANT:3(21)
 TRYING JUMP NP/DET ARC
  JUMP TRANS #6 FROM 7:NP/QUANT:3(21) TO 8:NP/DET:3(26)


SELECTED CONFIGS (8) FOR EXTENSION
PICKING UP CONFIG 8:NP/DET:3(26)
   STARTING AT 3:
    MONITORING [  NPR/ NPR/  ]
           {There are two PUSH NPR/ arcs from this state
           so two monitors are created, but only one
           configuration is set up.}
     SETTING UP CONFIG 9:NPR/:3(26)
    MONITORING [ NPR ]
    MONITORING [ NPR ]
    MONITORING [  N  ]
    MONITORING [  ADJ  ]
    MONITORING [  N  ]
    MONITORING [  V  ]
    MONITORING [  ADV  ]
 TRYING JUMP NP/HEAD ARC
  JUMP TRANS #7 FROM 8:NP/DET:3(26) TO 10:NP/HEAD:3(29)


SELECTED CONFIGS (10) FOR EXTENSION
PICKING UP CONFIG 10:NP/HEAD:3(29)
           {This is an example of the fallibility of using
           only context free tests on partial paths. The
           parser thinks it has successfully reached state
           NP/HEAD, while in fact this cannot be the case
           because no head noun has been discovered for
           the noun phrase. Thus it is incorrect to
           predict relative clauses at this point. This
           issue will be discussed in more detail in
           Chapter Seven.}
   STARTING AT 3:
    MONITORING [  R/ PP/ R/NIL  ]
     SETTING UP CONFIG 11:R/:3(29)
    MONITORING [ PREP ]
    MONITORING [ WHOSE ]
     PROPOSING "WHOSE"
    MONITORING [ WHO ]
     PROPOSING "WHO"
    MONITORING [ WHICH ]
     PROPOSING "WHICH"
    MONITORING [ THAT ]
     PROPOSING "THAT"
```

```
    MONITORING [ WHOM ]
     PROPOSING "WHOM"
     SETTING UP CONFIG 12:PP/:3(29)
    MONITORING [ PREP ]
     SETTING UP CONFIG 13:R/NIL:3(29)
    MONITORING [ THERE ]
     PROPOSING "THERE"
 TRYING POP ARC
  POP TRANS #8 FROM 10:NP/HEAD:3(29)
    EXECUTING PATH (3 5 6 7 8)
     BEGINNING AT TRANS 3, CONFIG 3
    DOING CAT ARC WITH WHQ FROM 3:NP/ TO 5:NP/ORD
    DOING JUMP ARC FROM 5:NP/ORD TO 7:NP/QUANT
    DOING JUMP ARC FROM 7:NP/QUANT TO 8:NP/DET
    DOING JUMP ARC FROM 8:NP/DET TO 10:NP/HEAD
     TEST FAILED
          {A question-determiner alone cannot be a
          complete noun phrase; although this is
          permitted by considering "what" as a QWORD as
          in transition #2.}


STARTING AN ISLAND
"REGISTRATION" TRYING CAT N ARC FROM NP/DET TO NP/DET
  CAT N TRANS #9 FROM 14:NP/DET:6(1) TO 15:NP/DET:19(4)
          {This arc is using "registration" as a noun
          modifier for some future head noun.}
   ENDING AT 6:
    MONITORING [  NPR/  ]
    MONITORING [  ADJ  ]
    MONITORING [  N  ]
    MONITORING [  V  ]
  JUMP TRANS #10 FROM 16:NP/QUANT:6(1) TO 14:NP/DET:6(1)
   ENDING AT 6:
    MONITORING [  QUANT/  ]
  JUMP TRANS #11 FROM 17:NP/ORD:6(1) TO 16:NP/QUANT:6(1)
   ENDING AT 6:
    MONITORING [  ORD  ]
    MONITORING [  QDET  ]
    MONITORING [  ONLY  ]
     PROPOSING "ONLY"
  JUMP TRANS #12 FROM 18:NP/ART:6(1) TO 17:NP/ORD:6(1)
   ENDING AT 6:
    MONITORING [  ART  ]
     NOTICING "THE"
    MONITORING [  QUANT  ]
    MONITORING [  POSS  ]
    MONITORING [  WHOSE  ]
     PROPOSING "WHOSE"
```

```
  JUMP TRANS #13 FROM 19:NP/ONLY:6(1) TO 18:NP/ART:6(1)
   ENDING AT 6:
    MONITORING [  ONLY  ]
     PROPOSING "ONLY"
  JUMP TRANS #14 FROM 20:NP/:6(1) TO 19:NP/ONLY:6(1)


"REGISTRATION" TRYING CAT N ARC FROM NP/DET TO NP/HEAD
  CAT N TRANS #15 FROM 14:NP/DET:6(1) TO 21:NP/HEAD:19(6)
           {This arcis using "registration" as the head
           noun of a noun phrase.}

SELECTED CONFIGS (21) FOR EXTENSION
PICKING UP CONFIG 21:NP/HEAD:19(6)
   STARTING AT 19:
    MONITORING [  R/ PP/ R/NIL  ]
     SETTING UP CONFIG 22:R/:19(6)
    MONITORING [ PREP ]
    MONITORING [ WHOSE ]
    MONITORING [ WHO ]
    MONITORING [ WHICH ]
    MONITORING [ THAT ]
    MONITORING [ WHOM ]
     SETTING UP CONFIG 23:PP/:19(6)
    MONITORING [ PREP ]
     SETTING UP CONFIG 24:R/NIL:19(6)
    MONITORING [ THERE ]
     NOTICING "FEE"
           {This notice is in response to the look-ahead
           test on the push arc to state R/NIL. Since
           "fee" can start a reduced relative clause, it
           is noticed, but there is not a specific monitor
           set up because the arc within the relative
           clause network which will actually process the
           word "fee" is not known.}
 TRYING POP ARC
  POP TRANS #16 FROM 21:NP/HEAD:19(6)
    EXECUTING PATH (14 13 12 11 10 15 16)
     BEGINNING AT TRANS 14, CONFIG 20
     TEST FAILED
           {The path failed because there is no determiner
           for "registration."}


PREDICTIONS:
NOTICING (4 FEE 19 23 155 0), SCORE -5
NOTICING (9 THE 4 6 103 0), SCORE 0
PROPOSING (ONLY WHOSE) ENDING AT 6
PROPOSING (ZERO NO POINT A WHOSE WHO WHICH THAT WHOM
 THERE) STARTING AT 3
```

```
MONITORING [  PREP  ] STARTING AT 19, SCORE 0
MONITORING [  WHOSE  ] STARTING AT 19, SCORE 5
MONITORING [  WHO  ] STARTING AT 19, SCORE 5
MONITORING [  WHICH  ] STARTING AT 19, SCORE 5
MONITORING [  THAT  ] STARTING AT 19, SCORE 5
MONITORING [  WHOM  ] STARTING AT 19, SCORE 5
MONITORING [  THERE  ] STARTING AT 19, SCORE 5
MONITORING [  ADJ  ] ENDING AT 6, SCORE 0
MONITORING [  N  ] ENDING AT 6, SCORE 0
MONITORING [  V  ] ENDING AT 6, SCORE 0
MONITORING [  ORD  ] ENDING AT 6, SCORE 0
MONITORING [  QDET  ] ENDING AT 6, SCORE 0
MONITORING [  ONLY  ] ENDING AT 6, SCORE 5
MONITORING [  ART  ] ENDING AT 6, SCORE 0
MONITORING [  QUANT  ] ENDING AT 6, SCORE 0
MONITORING [  POSS  ] ENDING AT 6, SCORE 0
MONITORING [  WHOSE  ] ENDING AT 6, SCORE 5
MONITORING [  MODAL  ] STARTING AT 3, SCORE 0
MONITORING [  V  ] STARTING AT 3, SCORE 0
MONITORING [  INTEGER  ] STARTING AT 3, SCORE 0
MONITORING [  ZERO  ] STARTING AT 3, SCORE 5
MONITORING [  NO  ] STARTING AT 3, SCORE 5
MONITORING [  POINT  ] STARTING AT 3, SCORE 5
MONITORING [  A  ] STARTING AT 3, SCORE 5
MONITORING [  NPR  ] STARTING AT 3, SCORE 0
MONITORING [  N  ] STARTING AT 3, SCORE 0
MONITORING [  ADJ  ] STARTING AT 3, SCORE 0
MONITORING [  V  ] STARTING AT 3, SCORE 0
MONITORING [  ADV  ] STARTING AT 3, SCORE 0
MONITORING [  PREP  ] STARTING AT 3, SCORE 0
MONITORING [  WHOSE  ] STARTING AT 3, SCORE 5
MONITORING [  WHO  ] STARTING AT 3, SCORE 5
MONITORING [  WHICH  ] STARTING AT 3, SCORE 5
MONITORING [  THAT  ] STARTING AT 3, SCORE 5
MONITORING [  WHOM  ] STARTING AT 3, SCORE 5
MONITORING [  THERE  ] STARTING AT 3, SCORE 5
PROPOSING (V N ADJ) FROM 3 TO 6
          {Proposals were made to fill the gap because
          there were monitors from both sides of a gap
          small enough to contain one word.}

FINISHED THEORY 1 WITH SYN SCORE 0

47 MONITORS SET
   18 WORDS
   18 CATEGORIES
   11 CONSTITS
   0 TESTS
```

```
13 PROPOSALS MADE
   10 WORDS
   3 CATEGORIES
2 NOTICES MADE
   2 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
0 CONSTITS MADE
   0 FOR OTHER THEORIES
   0 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {It took 11.9 seconds to process this theory.}
```

{Processing the proposals just made results, notably, in the detection of the word "other" between "what" and "registration", but the word match score is very low. Word matches for "is" and "are" from position 3 (next to "what") to position 4 are also found, but since they do not fill the gap, the event scores are low. The best event is that for the word "fee". Processing it is fairly uninteresting, since it completes no constituent, so we will omit the trace of that event. After it has been processed, however, the best event is that for the word "the" and the theory just created.}

```
SYNTAX PROCESSING EVENT FOR THEORY#2
  WITH NEW WORD (4 THE 6)
  TO GET NEW THEORY#3:
: WHAT 3   4 THE 6 REGISTRATION 19 FEE 23


"THE" TRYING (CAT ART --) FROM STATE NP/ONLY TO CONFIG 18
  CAT ART TRANS #26 FROM 32:NP/ONLY:4(3) TO 18:NP/ART:6(6)
   ENDING AT 4:
    MONITORING [  ONLY  ]
     PROPOSING "ONLY"
  JUMP TRANS #27 FROM 33:NP/:4(1) TO 32:NP/ONLY:4(3)
    EXECUTING PATH (27 26 12 11 10 9 22 25)
     BEGINNING AT TRANS 22, CONFIG 15
```

```
****
    MADE #1 FROM 4 TO 23:
    NP DET ART THE
       ADJ NP N REGISTRATION
              NU SG
       N FEE
       FEATS NU SG
****
```

{The format of this noun phrase is slightly different from that in the previous examples because the structure building action for noun phrases in SPEECHGRAMMAR is different from that in MINIGRAMMAR>

There are many places in the SPEECHGRAMMAR which push for noun phrases, and since there were no monitors in the WFST which can use this constituent, all of them must be tried, resulting in a number of predictions and notices.}

```
SYN WEIGHT + SEM WT = 15 + 0 = 15


NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #28 FROM 34:FOR/FOR:4(1) TO 35:TO/:23(10)
   ENDING AT 4:
    MONITORING [  FOR  ]
     PROPOSING "FOR"

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #29 FROM 36:PP/PREP:4(1) TO
 37:PP/NP:23(10)
   ENDING AT 4:
    MONITORING [  PREP  ]

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #30 FROM 38:R/NIL:4(1) TO 39:S/NP:23(9)

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #31 FROM 40:R/WH:4(1) TO 39:S/NP:23(9)
   ENDING AT 4:
    MONITORING [  R/WHOSE  ]
    MONITORING [  WHICH THAT WHO WHOM WHICH WHOM  ]
     PROPOSING "WHICH"
     PROPOSING "THAT"
     PROPOSING "WHO"
     PROPOSING "WHOM"
     PROPOSING "WHICH"
     PROPOSING "WHOM"
```

{There are two arcs entering state R/WH which use the words "which" and "whom". There is a check made to see that duplicate proposals are not actually communicated to the control

```
          component,  although  they  appear  to  be
          duplicated in the trace.}

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #32 FROM 41:S/DCL:4(1) TO 39:S/NP:23(9)
  JUMP TRANS #33 FROM 42:S/:4(1) TO 41:S/DCL:4(1)
   ENDING AT 4:
    MONITORING [  PP/  ]

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #34 FROM 43:S/NO-SUBJ:4(1) TO
      44:VP/V:23(9)
  JUMP TRANS #35 FROM 45:S/AUX:4(1) TO 43:S/NO-SUBJ:4(1)
   ENDING AT 4:
    MONITORING [  MODAL  ]
    MONITORING [  NEG  ]
    MONITORING [  V  ]
     NOTICING "IS"
     NOTICING "ARE"

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #36 FROM 46:S/Q:4(1) TO 39:S/NP:23(9)
   ENDING AT 4:
    MONITORING [  QADV  ]

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #37 FROM 47:VP/HEAD:4(1) TO
 48:VP/NP:23(9)
   ENDING AT 4:
    MONITORING [  PARTICLE  ]
    MONITORING [  V  ]
  JUMP TRANS #38 FROM 49:VP/V:4(1) TO 47:VP/HEAD:4(1)
   ENDING AT 4:
    MONITORING [  NP/ NP/  ]
    MONITORING [  V  ]
     NOTICING "IS"
     NOTICING "ARE"
    MONITORING [  V  ]
     NOTICING "IS"
     NOTICING "ARE"
    MONITORING [  ADV  ]
    MONITORING [  V  ]
          {The words "is" and "are"  failed  the  context
          free  test  on  the  arc  causing this monitor,
          hence they are not noticed.}
  JUMP TRANS #39 FROM 45:S/AUX:4(1) TO 49:VP/V:4(1)
  JUMP TRANS #40 FROM 43:S/NO-SUBJ:4(1) TO 49:VP/V:4(1)
  JUMP TRANS #41 FROM 43:S/NO-SUBJ:4(1) TO 49:VP/V:4(1)
```

```
  JUMP TRANS #42 FROM 50:S/THERE:4(1) TO 49:VP/V:4(1)
   ENDING AT 4:
    MONITORING [  THERE  ]
     PROPOSING "THERE"

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #43 FROM 51:VP/NP:4(1) TO 52:VP/VP:23(9)
   ENDING AT 4:
    MONITORING [  NP/  ]
  JUMP TRANS #44 FROM 47:VP/HEAD:4(1) TO 51:VP/NP:4(1)

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #45 FROM 49:VP/V:4(1) TO 44:VP/V:23(9)

          {The creation of transition #27 completed 3
          paths.  The first two have been executed,
          resulting respectively in failure and the
          completion  of a constituent with all the
          processing that entails.  Now the third path is
          still pending and is about to be executed.}
    EXECUTING PATH (27 26 12 11 10 15 16)
     BEGINNING AT TRANS 15, CONFIG 14
    DOING CAT ARC WITH REGISTRATION FROM 14:NP/DET TO
       21:NP/HEAD
    DOING POP ARC FROM 21:NP/HEAD
****
    MADE #2 FROM 4 TO 19:
    NP DET ART THE
       N REGISTRATION
       FEATS NU SG
****
          {This constituent can now satisfy the monitors
          set  by the discovery of the larger one,
          resulting in the creation of many new
          transitions but no new predictions.}
SYN WEIGHT + SEM WT = 10 + 0 = 10

NP/ WAS PUSHED FOR AT CONFIG 34
  PUSH NP/ TRANS #46 FROM 34:FOR/FOR:4(1) TO 53:TO/:19(8)

NP/ WAS PUSHED FOR AT CONFIG 36
 PUSH NP/ TRANS #47 FROM 36:PP/PREP:4(1) TO 54:PP/NP:19(8)

NP/ WAS PUSHED FOR AT CONFIG 38
  PUSH NP/ TRANS #48 FROM 38:R/NIL:4(1) TO 55:S/NP:19(7)

NP/ WAS PUSHED FOR AT CONFIG 40
  PUSH NP/ TRANS #49 FROM 40:R/WH:4(1) TO 55:S/NP:19(7)

NP/ WAS PUSHED FOR AT CONFIG 41
  PUSH NP/ TRANS #50 FROM 41:S/DCL:4(1) TO 55:S/NP:19(7)
```

```
NP/ WAS PUSHED FOR AT CONFIG 43
  PUSH NP/ TRANS #51 FROM 43:S/NO-SUBJ:4(1) TO
       56:VP/V:19(7)

NP/ WAS PUSHED FOR AT CONFIG 46
  PUSH NP/ TRANS #52 FROM 46:S/Q:4(1) TO 55:S/NP:19(8)

NP/ WAS PUSHED FOR AT CONFIG 47
  PUSH NP/ TRANS #53 FROM 47:VP/HEAD:4(1) TO
       57:VP/NP:19(8)

NP/ WAS PUSHED FOR AT CONFIG 51
  PUSH NP/ TRANS #54 FROM 51:VP/NP:4(1) TO 58:VP/VP:19(7)

NP/ WAS PUSHED FOR AT CONFIG 49
  PUSH NP/ TRANS #55 FROM 49:VP/V:4(1) TO 56:VP/V:19(8)


SELECTED CONFIGS (55 54 39 37) FOR EXTENSION
          {Because these are the maximally scoring
          configurations from the large pool of
          possibilities.}
PICKING UP CONFIG 55:S/NP:19(8) WITH WORD FEE
 TRYING POP ARC
  POP TRANS #56 FROM 55:S/NP:19(8)
    EXECUTING PATH (48 56)
     BEGINNING AT TRANS 48, CONFIG 38
     TEST FAILED
    EXECUTING PATH (33 50 56)
     BEGINNING AT TRANS 33, CONFIG 42
****
    MADE #3 FROM 4 TO 19:
    S NPU
      NP DET ART THE
         ADJ NP N REGISTRATION
                NU SG
         N FEE
         FEATS NU SG
    WITH FEATURES (NPU)
****
```

{Here is an example of a constituent which has features attached to it. The feature NPU can be tested by the semantic component to determine that the constituent is a noun phrase utterance. If necessary, it could also be tested on a PUSH S/ arc in the grammar, since there are some times, e.g. during the construction of a sentential complement, when an embedded sentence must contain a verb.}

```
SYN WEIGHT + SEM WT = 10 + 0 = 10
           {No arcs in this grammar push for noun phrase
           utterances, so this constituent is not used
           further.}

PICKING UP CONFIG 54:PP/NP:19(8) WITH WORD FEE
 TRYING POP ARC
  POP TRANS #57 FROM 54:PP/NP:19(8)

PICKING UP CONFIG 39:S/NP:23(9)
 TRYING POP ARC
  POP TRANS #58 FROM 39:S/NP:23(9)
    EXECUTING PATH (30 58)
     BEGINNING AT TRANS 30, CONFIG 38
     TEST FAILED
    EXECUTING PATH (33 32 58)
     BEGINNING AT TRANS 32, CONFIG 41
****
    MADE #4 FROM 4 TO 23:
    S NPU
      NP ADJ NP N REGISTRATION
                 NU SG
         DET ART THE
         N FEE
         FEATS NU SG
    WITH FEATURES (NPU)
****
SYN WEIGHT + SEM WT = 15 + 0 = 15

PICKING UP CONFIG 37:PP/NP:23(10)
 TRYING POP ARC
  POP TRANS #59 FROM 37:PP/NP:23(10)


PREDICTIONS:
NOTICING (19 IS 3 4 -79 0), SCORE 10
NOTICING (20 ARE 3 4 -128 0), SCORE 10
PROPOSING (ONLY FOR WHICH THAT WHO WHOM THERE) ENDING AT 4
MONITORING [  ONLY  ] ENDING AT 4, SCORE 15
MONITORING [  FOR  ] ENDING AT 4, SCORE 15
MONITORING [  PREP  ] ENDING AT 4, SCORE 10
MONITORING [  WHICH  ] ENDING AT 4, SCORE 15
MONITORING [  THAT  ] ENDING AT 4, SCORE 15
MONITORING [  WHO  ] ENDING AT 4, SCORE 15
MONITORING [  WHOM  ] ENDING AT 4, SCORE 15
MONITORING [  MODAL  ] ENDING AT 4, SCORE 10
MONITORING [  NEG  ] ENDING AT 4, SCORE 10
MONITORING [  V  ] ENDING AT 4, SCORE 10
MONITORING [  QADV  ] ENDING AT 4, SCORE 10
MONITORING [  PARTICLE  ] ENDING AT 4, SCORE 10
MONITORING [  V  ] ENDING AT 4, SCORE 10
MONITORING [  V  ] ENDING AT 4, SCORE 10
```

```
MONITORING [  V  ] ENDING AT 4, SCORE 10
MONITORING [  ADV  ] ENDING AT 4, SCORE 10
MONITORING [  V  ] ENDING AT 4, SCORE 10
MONITORING [  THERE  ] ENDING AT 4, SCORE 15
MONITORING [  MODAL  ] STARTING AT 3, SCORE 0
MONITORING [  V  ] STARTING AT 3, SCORE 0
MONITORING [  INTEGER  ] STARTING AT 3, SCORE 0
MONITORING [  ZERO  ] STARTING AT 3, SCORE 5
MONITORING [  NO  ] STARTING AT 3, SCORE 5
MONITORING [  POINT  ] STARTING AT 3, SCORE 5
MONITORING [  A  ] STARTING AT 3, SCORE 5
MONITORING [  NPR  ] STARTING AT 3, SCORE 0
MONITORING [  N  ] STARTING AT 3, SCORE 0
MONITORING [  ADJ  ] STARTING AT 3, SCORE 0
MONITORING [  V  ] STARTING AT 3, SCORE 0
MONITORING [  ADV  ] STARTING AT 3, SCORE 0
MONITORING [  PREP  ] STARTING AT 3, SCORE 0
MONITORING [  WHOSE  ] STARTING AT 3, SCORE 5
MONITORING [  WHO  ] STARTING AT 3, SCORE 5
MONITORING [  WHICH  ] STARTING AT 3, SCORE 5
MONITORING [  THAT  ] STARTING AT 3, SCORE 5
MONITORING [  WHOM  ] STARTING AT 3, SCORE 5
MONITORING [  THERE  ] STARTING AT 3, SCORE 5
PROPOSING (MODAL) FROM 3 TO 4
PROPOSING (MODAL PREP) STARTING AT 3
PROPOSING (PREP MODAL NEG QADV) ENDING AT 4

CREATING THEORY 3:
0 WHAT 3   4 THE 6 REGISTRATION 19 FEE 23
 WITH SYN SCORE 15

42 MONITORS SET
   17 WORDS
   20 CATEGORIES
   5 CONSTITS
   0 TESTS
11 PROPOSALS MADE
   7 WORDS
   4 CATEGORIES
2 NOTICES MADE
   2 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
4 CONSTITS MADE
   0 FOR OTHER THEORIES
   4 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
```

{This event took 34.5 seconds, largely because of the extensive bottom up processing necessitated by the discovery of the noun phrases which were not monitored for.}

{Processing the proposals from this theory results in the best event being the one for "is" in the last gap. The word "are" also fills the gap, but the lower lexical score prevents the event for it from surfacing. If it were syntactically processed, however, no new theory would be created since the completed string would be ungrammatical.}

```
SYNTAX PROCESSING EVENT FOR THEORY#3
  WITH NEW WORD (3 IS 4)
  TO GET NEW THEORY#4:
0 WHAT 3 IS 4 THE 6 REGISTRATION 19 FEE 23

"IS" TRYING (CAT V --) FROM CONFIG 4
  CAT V TRANS #60 FROM 4:S/NP:3(11) TO 45:S/AUX:4(16)
```

{This transition does not immediately complete any paths, so the best scoring configurations of the theory are tried.}

```
SELECTED CONFIGS (31) FOR EXTENSION
PICKING UP CONFIG 31:NP/DET:23(36)
 TRYING JUMP NP/HEAD ARC
  JUMP TRANS #61 FROM 31:NP/DET:23(36) TO
       30:NP/HEAD:23(39)


SELECTED CONFIGS (52 48 44) FOR EXTENSION
PICKING UP CONFIG 52:VP/VP:23(9)
 TRYING JUMP S/VP ARC
  JUMP TRANS #62 FROM 52:VP/VP:23(9) TO 59:S/VP:23(12)
PICKING UP CONFIG 48:VP/NP:23(9)
 TRYING JUMP VP/VP ARC
  JUMP TRANS #63 FROM 48:VP/NP:23(9) TO 52:VP/VP:23(11)
PICKING UP CONFIG 44:VP/V:23(9)
 TRYING JUMP VP/HEAD ARC
  JUMP TRANS #64 FROM 44:VP/V:23(9) TO 60:VP/HEAD:23(13)


SELECTED CONFIGS (60) FOR EXTENSION
PICKING UP CONFIG 60:VP/HEAD:23(13)
 TRYING JUMP VP/NP ARC
  JUMP TRANS #65 FROM 60:VP/HEAD:23(13) TO 48:VP/NP:23(16)
```

```
SELECTED CONFICS (59) FOR EXTENSION
PICKING UP CONFIG 59:S/VP:23(12)
 TRYING JUMP S/S ARC
  JUMP TRANS #66 FROM 59:S/VP:23(12) TO 61:S/S:23(14)

SELECTED CONFIGS (61) FOR EXTENSION
PICKING UP CONFIG 61:S/S:23(14)
 TRYING POP ARC
  POP TRANS #67 FROM 61:S/S:23(14)
    EXECUTING PATH (1 2 60 35 34 64 65 63 62 66 67)
     BEGINNING AT TRANS 34, CONFIG 43
****
    MADE #5 FROM 0 TO 23:
    S Q
      SUBJ NP DET ART THE
              ADJ NP N REGISTRATION
                      NU SG
              N FEE
              FEATS NU SG
      AUX TNS PRESENT
          VOICE ACTIVE
      VP V BE
         OBJ NP N WHAT
                FEATS NU SG/PL
****
           {This is the complete parse of the utterance,
           but SPARSER continues the operations it has
           pending before returning to Control.}
   NO SEMANTICS FOR HEAD
           {This is a comment from the semantic component
           indicating that it cannot currently interpret
           the construction.}
SYN WEIGHT + SEM WT = 25 + 0 = 25

S/ WAS NEVER PUSHED FOR
  PUSH S/ TRANS #68 FROM 62:COMPL/NTYPE:0(1) TO
       63:COMPL/S:23(15)

S/ WAS NEVER PUSHED FOR
  PUSH S/ TRANS #69 FROM 64:S/THEN:0(1) TO
       65:S/IFTHEN:23(15)

S/ WAS NEVER PUSHED FOR
  PUSH S/ TRANS #70 FROM 66:VP/HEAD:0(1) TO
       52:VP/VP:23(13)
  JUMP TRANS #71 FROM 67:VP/V:0(1) TO 66:VP/HEAD:0(1)
  JUMP TRANS #72 FROM 68:S/AUX:0(1) TO 67:VP/V:0(1)
  JUMP TRANS #73 FROM 69:S/NO-SUBJ:0(1) TO 67:VP/V:0(1)
  JUMP TRANS #74 FROM 68:S/AUX:0(1) TO 69:S/NO-SUBJ:0(1)
  JUMP TRANS #75 FROM 69:S/NO-SUBJ:0(1) TO 67:VP/V:0(1)
  JUMP TRANS #76 FROM 70:S/THERE:0(1) TO 67:VP/V:0(1
```

```
          {One of the pending operations is to check the
          other arcs which caused monitors for the verb
          "is".}
"IS" TRYING (CAT V --) FROM STATE S/NP TO CONFIG 45


"IS" TRYING (CAT V --) FROM STATE FOR/TO TO CONFIG 49
  CAT V TRANS #77 FROM 71:FOR/TO:3(5) TO 49:VP/V:4(20)

"IS" TRYING (CAT V --) FROM STATE VP/V TO CONFIG 49
  CAT V TRANS #78 FROM 72:VP/V:3(5) TO 49:VP/V:4(20)
  JUMP TRANS #79 FROM 73:S/AUX:3(1) TO 72:VP/V:3(5)
  JUMP TRANS #80 FROM 74:S/NO-SUBJ:3(1) TO 72:VP/V:3(5)
  JUMP TRANS #81 FROM 73:S/AUX:3(1) TO 74:S/NO-SUBJ:3(1)
  JUMP TRANS #82 FROM 74:S/NO-SUBJ:3(1) TO 72:VP/V:3(5)
  JUMP TRANS #83 FROM 75:S/THERE:3(1) TO 72:VP/V:3(5)


CREATING THEORY 4:
0 WHAT 3 IS 4 THE 6 REGISTRATION 19 FEE 23
 WITH SYN SCORE 15


2 MONITORS SET
   0 WORDS
   0 CATEGORIES
   2 CONSTITS
   0 TESTS
0 PROPOSALS MADE
   0 WORDS
   0 CATEGORIES
0 NOTICES MADE
   0 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
1 CONSTITS MADE
   0 FOR OTHER THEORIES
   5 FOR THIS THEORY
   1 COMPLETE PARSINGS FOUND
          {This processing took 34.45 seconds.}
```

This example was run with a very simple, mechanical control structure. After the processing of the initial theory, the proposals which had been made by SPARSR were processed by the lexical retrieval component and the results added to the word lattice -- a process which can

set off monitors and result in the creation of event notices. The events are scored by a combination of the monitor score assigned by SPARSER and the lexical score assigned by the word match component. In this sentence, syntax and lexical score alone were sufficient to make the best scoring event at each step be one which resulted in a correct extension of the theory.

6.5 EXAMPLE 4

We now show how the same utterance used in the previous example can be recognized when different theories are created and when events and theories are processed in a different order from that in Example 3. Suppose that after the initial scan of the utterance (resulting in the word lattice shown at the beginning of Section 6.4) the semantic component created two theories, one for the words "what" and "fee" and the other for the words "what" and "registration". Let us see what happens in SPARSER when we begin by processing these two theories in sequence.

SPARSER PROCESSING THEORY 1:
0 WHAT 3 19 FEE 23
{The processing of this theory is very similar to that of the first theory in the previous example, and will not be commented upon here.

The purpose in showing it is to provide a map, part of which the next call to SPARSER will trace.}

```
STARTING AN ISLAND
STARTING AT LEFT END OF SENTENCE

SELECTED CONFIGS (1) FOR EXTENSION
PICKING UP CONFIG 1:S/:0(1) WITH WORD WHAT
 TRYING PUSH PP/ ARC
 TRYING JUMP S/Q ARC
  JUMP TRANS #1 FROM 1:S/:0(1) TO 2:S/Q:0(6)
 TRYING WRD IF ARC
 TRYING JUMP S/IMP ARC
 TRYING JUMP S/DCL ARC

SELECTED CONFIGS (2) FOR EXTENSION
PICKING UP CONFIG 2:S/Q:0(6) WITH WORD WHAT
 TRYING PUSH NP/ ARC
      MONITORING  [ NP/ ]
         SETTING UP CONFIG 3:NP/:0(11)
 TRYING WRD HOW ARC
 TRYING CAT QWORD ARC
  CAT QWORD TRANS #2 FROM 2:S/Q:0(6) TO 4:S/NP:3(11)
 TRYING CAT QADV ARC
 TRYING JUMP S/NP ARC

SELECTED CONFIGS (3 4) FOR EXTENSION

PICKING UP CONFIG 3:NP/:0(11) WITH WORD WHAT
 TRYING WRD ONLY ARC
 TRYING CAT QDET ARC
  CAT QDET TRANS #3 FROM 3:NP/:0(11) TO 5:NP/ORD:3(16)
 TRYING PUSH DATE/ ARC
 TRYING TST ARC
 TRYING JUMP NP/ONLY ARC

PICKING UP CONFIG 4:S/NP:3(11)
   STARTING AT 3:
    MONITORING [  MODAL  ]
    MONITORING [  V  ]
 TRYING POP ARC
  POP TRANS #4 FROM 4:S/NP:3(11)
    EXECUTING PATH (1 2 4)
     BEGINNING AT TRANS 1, CONFIG 1
    DOING JUMP ARC WITH WHAT FROM 1:S/ TO 2:S/Q
    DOING CAT ARC WITH WHAT FROM 2:S/Q TO 4:S/NP
```

```
    DOING POP ARC FROM 4:S/NP
     TEST FAILED

SELECTED CONFIGS (5) FOR EXTENSION
PICKING UP CONFIG 5:NP/ORD:3(16)
   STARTING AT 3:
    MONITORING [  QUANT/  ]
     SETTING UP CONFIG 6:QUANT/:3(16)
    MONITORING [ INTEGER ]
 TRYING JUMP NP/QUANT ARC
  JUMP TRANS #5 FROM 5:NP/ORD:3(16) TO 7:NP/QUANT:3(21)

SELECTED CONFIGS (7) FOR EXTENSION
PICKING UP CONFIG 7:NP/QUANT:3(21)
 TRYING JUMP NP/DET ARC
  JUMP TRANS #6 FROM 7:NP/QUANT:3(21) TO 8:NP/DET:3(26)

SELECTED CONFIGS (8) FOR EXTENSION
PICKING UP CONFIG 8:NP/DET:3(26)
   STARTING AT 3:
    MONITORING [  NPR/ NPR/  ]
     SETTING UP CONFIG 9:NPR/:3(26)
    MONITORING [ NPR ]
     SETTING UP CONFIG 9:NPR/:3(26)
    MONITORING [ NPR ]
    MONITORING [  N  ]
    MONITORING [  ADJ  ]
    MONITORING [  N  ]
    MONITORING [  V  ]
    MONITORING [  ADV  ]
 TRYING JUMP NP/HEAD ARC
  JUMP TRANS #7 FROM 8:NP/DET:3(26) TO 10:NP/HEAD:3(29)

SELECTED CONFIGS (10) FOR EXTENSION
PICKING UP CONFIG 10:NP/HEAD:3(29)
   STARTING AT 3:
    MONITORING [  R/ PP/ R/NIL  ]
     SETTING UP CONFIG 11:R/:3(29)
    MONITORING [ PREP ]
    MONITORING [ WHOSE ]
     PROPOSING "WHOSE"
    MONITORING [ WHO ]
     PROPOSING "WHO"
    MONITORING [ WHICH ]
     PROPOSING "WHICH"
    MONITORING [ THAT ]
     PROPOSING "THAT"
    MONITORING [ WHOM ]
     PROPOSING "WHOM"
     SETTING UP CONFIG 12:PP/:3(29)
    MONITORING [ PREP ]
     SETTING UP CONFIG 13:R/NIL:3(29)
```

```
    MONITORING [ THERE ]
     PROPOSING "THERE"
 TRYING POP ARC
  POP TRANS #8 FROM 10:NP/HEAD:3(29)
    EXECUTING PATH (3 5 6 7 8)
     BEGINNING AT TRANS 3, CONFIG 3
    DOING CAT ARC WITH WHQ FROM 3:NP/ TO 5:NP/ORD
    DOING JUMP ARC FROM 5:NP/ORD TO 7:NP/QUANT
    DOING JUMP ARC FROM 7:NP/QUANT TO 8:NP/DET
    DOING JUMP ARC FROM 8:NP/DET TO 10:NP/HEAD
     TEST FAILED


STARTING AN ISLAND
"FEE" TRYING CAT N ARC FROM NP/DET TO NP/DET
  CAT N TRANS #9 FROM 14:NP/DET:19(1) TO 15:NP/DET:23(4)
   ENDING AT 19:
    MONITORING [  NPR/  ]
    MONITORING [  ADJ  ]
    MONITORING [  N  ]
     NOTICING "REGISTRATION"
     NOTICING "REGISTRATION"
           {There are two instances of the word
           "registration" in the word lattice, hence two
           notices are created.}
    MONITORING [  V  ]
  JUMP TRANS #10 FROM 16:NP/QUANT:19(1) TO 14:NP/DET:19(1)
   ENDING AT 19:
    MONITORING [  QUANT/  ]
  JUMP TRANS #11 FROM 17:NP/ORD:19(1) TO 16:NP/QUANT:19(1)
   ENDING AT 19:
    MONITORING [  ORD  ]
    MONITORING [  QDET  ]
    MONITORING [  ONLY  ]
     PROPOSING "ONLY"
  JUMP TRANS #12 FROM 18:NP/ART:19(1) TO 17:NP/ORD:19(1)
   ENDING AT 19:
    MONITORING [  ART  ]
    MONITORING [  QUANT  ]
    MONITORING [  POSS  ]
    MONITORING [  WHOSE  ]
     PROPOSING "WHOSE"
  JUMP TRANS #13 FROM 19:NP/ONLY:19(1) TO 18:NP/ART:19(1)
   ENDING AT 19:
    MONITORING [  ONLY  ]
     PROPOSING "ONLY"
  JUMP TRANS #14 FROM 20:NP/:19(1) TO 19:NP/ONLY:19(1)
```

```
"FEE" TRYING CAT N ARC FROM NP/DET TO NP/HEAD
  CAT N TRANS #15 FROM 14:NP/DET:19(1) TO 21:NP/HEAD:23(6)

SELECTED CONFIGS (21) FOR EXTENSION
PICKING UP CONFIG 21:NP/HEAD:23(6)
 TRYING POP ARC
  POP TRANS #16 FROM 21:NP/HEAD:23(6)
    EXECUTING PATH (14 13 12 11 10 15 16)
     BEGINNING AT TRANS 14, CONFIG 20
     TEST FAILED


PREDICTIONS:
NOTICING (1 REGISTRATION 6 19 277 0), SCORE -5
NOTICING (10 REGISTRATION 7 19 103 0), SCORE -5
PROPOSING (ONLY WHOSE) ENDING AT 19
PROPOSING (WHOSE WHO WHICH THAT WHOM THERE) STARTING AT 3
MONITORING [  ADJ  ] ENDING AT 19, SCORE 0
MONITORING [  N  ] ENDING AT 19, SCORE 0
MONITORING [  V  ] ENDING AT 19, SCORE 0
MONITORING [  ORD  ] ENDING AT 19, SCORE 0
MONITORING [  QDET  ] ENDING AT 19, SCORE 0
MONITORING [  ONLY  ] ENDING AT 19, SCORE 5
MONITORING [  ART  ] ENDING AT 19, SCORE 0
MONITORING [  QUANT  ] ENDING AT 19, SCORE 0
MONITORING [  POSS  ] ENDING AT 19, SCORE 0
MONITORING [  WHOSE  ] ENDING AT 19, SCORE 5
MONITORING [  MODAL  ] STARTING AT 3, SCORE 0
MONITORING [  V  ] STARTING AT 3, SCORE 0
MONITORING [  INTEGER  ] STARTING AT 3, SCORE 0
MONITORING [  NPR  ] STARTING AT 3, SCORE 0
MONITORING [  N  ] STARTING AT 3, SCORE 0
MONITORING [  ADJ  ] STARTING AT 3, SCORE 0
MONITORING [  V  ] STARTING AT 3, SCORE 0
MONITORING [  ADV  ] STARTING AT 3, SCORE 0
MONITORING [  PREP  ] STARTING AT 3, SCORE 0
MONITORING [  WHOSE  ] STARTING AT 3, SCORE 5
MONITORING [  WHO  ] STARTING AT 3, SCORE 5
MONITORING [  WHICH  ] STARTING AT 3, SCORE 5
MONITORING [  THAT  ] STARTING AT 3, SCORE 5
MONITORING [  WHOM  ] STARTING AT 3, SCORE 5
MONITORING [  THERE  ] STARTING AT 3, SCORE 5

FINISHED THEORY 1 WITH SYN SCORE 0

33 MONITORS SET
   8 WORDS
   17 CATEGORIES
   8 CONSTITS
   0 TESTS
```

```
6 PROPOSALS MADE
   6 WORDS
   0 CATEGORIES
2 NOTICES MADE
   3 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
0 CONSTITS MADE
   0 FOR OTHER THEORIES
   0 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {This took 12.5 seconds.}


          {Now we will process the second theory.}
SPARSER PROCESSING THEORY 2:
0 WHAT 3   6 REGISTRATION 19  23

STARTING AN ISLAND
STARTING AT LEFT END OF SENTENCE

SELECTED CONFIGS (1) FOR EXTENSION
          {Upon picking up this configuration to extend
          it, SPARSER finds the transitions which were
          created during the processing of the word
          "what" by the previous theory. It "traces"
          them all, that is, it does not recreate them
          but simply puts the transition numbers on a
          list which will form part of the syntactic
          information associated with the current theory.
          The tracing process also involves the cr of
          monitors (and notices, where applicable) for
          constituents along the path. These monitors
          and notices must be remade, since the previous
          monitors will activate only the previous
          theory.
          Due to the recursive nature of the tracing
          process, the transitions are not necessarily
          followed in the same order that they were
          originally created, nor are the monitors made
          in exactly the same order.
          Notice that the many arcs which were tried but
          which did not result in the creation of
          transitions in the previous theory are not
          retried here.}
PICKING UP CONFIG 1:S/:0(1) WITH WORD WHAT
  TRACING JUMP S/Q TRANS 1 FROM 1:S/:0(1) TO 2:S/Q:0(6)
     MONITORING  [ NP/ ]
  TRACING CAT QWORD TRANS 2 USING "WHAT" FROM 2:S/Q:0(6)
       TO 4:S/NP:3(11)
```

```
 STARTING AT 3:
  MONITORING [  MODAL  ]
  MONITORING [  V  ]
TRACING POP TRANS 4 FROM 4:S/NP:3(11)
TRACING CAT QDET TRANS 3 USING "WHQ" FROM 3:NP/:0(11)
     TO 5:NP/ORD:3(16)
 STARTING AT 3:
  MONITORING [  QUANT/  ]
   SETTING UP CONFIG 6:QUANT/:3(16)
        {This does not mean that  configuration  6  was
        just  created.  Since it already existed in the
        map, having been created during the  processing
        of   the  previous  theory,  the  configuration
        number  is  merely   put   on   the   list   of
        configurations in the current theory.}
  MONITORING [ INTEGER ]
TRACING JUMP NP/QUANT TRANS 5 FROM 5:NP/ORD:3(16) TO
     7:NP/QUANT:3(21)
TRACING JUMP NP/DET TRANS 6 FROM 7:NP/QUANT:3(21) TO
     8:NP/DET:3(26)
 STARTING AT 3:
  MONITORING [  NPR/ NPR/  ]
   SETTING UP CONFIG 9:NPR/:3(26)
  MONITORING [ NPR ]
   SETTING UP CONFIG 9:NPR/:3(26)
  MONITORING [ NPR ]
  MONITORING [  N  ]
  MONITORING [  ADJ  ]
  MONITORING [  N  ]
  MONITORING [  V  ]
  MONITORING [  ADV  ]
TRACING JUMP NP/HEAD TRANS 7 FROM 8:NP/DET:3(26) TO
     10:NP/HEAD:3(29)
 STARTING AT 3:
  MONITORING [  R/ PP/ R/NIL  ]
   SETTING UP CONFIG 11:R/:3(29)
  MONITORING [ PREP ]
  MONITORING [ WHOSE ]
   NOTICING "WHOSE"
   NOTICING "WHO"
  MONITORING [ WHO ]
  MONITORING [ WHICH ]
  MONITORING [ THAT ]
  MONITORING [ WHOM ]
   SETTING UP CONFIG 12:PP/:3(29)
  MONITORING [ PREP ]
   SETTING UP CONFIG 13:R/NIL:3(29)
  MONITORING [ THERE ]
   NOTICING "WHOSE"
   NOTICING "WHO"
```

{No proposals were made here because proposals are not theory dependent; that is, the word proposals which were made during the processing of the previous theory resulted in some words being placed in the word lattice which were noticed here. Remaking the proposals would not lead to the discovery of any new information.}

```
  TRACING POP TRANS 8 FROM 10:NP/HEAD:3(29)
```

{The processing of the island for "registration" is identical to that in the last example, so the remainder of the trace will be omitted. The total processing took 12.2 seconds.}

{Let us now process the event which adds the word "the" to the theory just processed. This will result in the creation of a constituent event.}

```
SYNTAX PROCESSING EVENT FOR THEORY#2
  WITH NEW WORD (4 THE 6)
  TO GET NEW THEORY#3:
0 WHAT 3   4 THE 6 REGISTRATION 19  23

"THE" TRYING (CAT ART --) FROM STATE NP/ONLY TO CONFIG 25
  CAT ART TRANS #25 FROM 32:NP/ONLY:4(3) TO 25:NP/ART:6(6)
   ENDING AT 4:
    MONITORING [  ONLY  ]
     PROPOSING "ONLY"
  JUMP TRANS #26 FROM 33:NP/:4(1) TO 32:NP/ONLY:4(3)
    EXECUTING PATH (26 25 20 19 18 17 15 16)
     BEGINNING AT TRANS 26, CONFIG 33
****
    MADE #1 FROM 4 TO 23:
    NP DET ART THE
       ADJ NP N REGISTRATION
              NU SG
       N FEE
       FEATS NU SG
****
          NOTIFYING THEORY 3 ABOUT CONSTITUENT #1
```

{This constituent cannot be used immediately by this theory because it contains a word ("fee") which is not in the theory. Therefore a notice is sent to Control which may be turned into an event at some later time. Nothing further is done with this constituent at this time, i.e., no transitions using it are created. It is, however, placed in the WFST for later use.}

```
    EXECUTING PATH (26 25 20 19 18 23 24)
     BEGINNING AT TRANS 23, CONFIG 22
           {The creation of transition #26 completes
           another path.}
****
    MADE #2 FROM 4 TO 19:
    NP DET ART THE
       N REGISTRATION
       FEATS NU SG
****
SYN WEIGHT + SEM WT = 10 + 0 = 10

           {This constituent is completely consistent with
           the current theory, that is, it is composed
           only of word matches already in the theory, and
           there are no monitors in the WFST for it, so it
           is processed bottom up as we have seen before.}
NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #27 FROM 34:FOR/FOR:4(1) TO 35:TO/:19(7)
   ENDING AT 4:
    MONITORING [  FOR  ]
     PROPOSING "FOR"

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #28 FROM 36:PP/PREP:4(1) TO
       37:PP/NP:19(7)
   ENDING AT 4:
    MONITORING [  PREP  ]

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #29 FROM 38:R/NIL:4(1) TO 39:S/NP:19(6)

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #30 FROM 40:R/WH:4(1) TO 39:S/NP:19(6)
   ENDING AT 4:
    MONITORING [  R/WHOSE  ]
    MONITORING [  WHICH THAT WHO WHOM WHICH WHOM  ]
     NOTICING "WHO"
     PROPOSING "WHOM"
     PROPOSING "WHICH"
     PROPOSING "WHOM"

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #31 FROM 41:S/DCL:4(1) TO 39:S/NP:19(6)
  JUMP TRANS #32 FROM 42:S/:4(1) TO 41:S/DCL:4(1)
   ENDING AT 4:
    MONITORING [  PP/  ]

NP/ WAS NEVER PUSHED FOR
  PUSH NP/ TRANS #33 FROM 43:S/NO-SUBJ:4(1) TO
       44:VP/V:19(6)
```

```
   JUMP TRANS #34 FROM 45:S/AUX:4(1) TO 43:S/NO-SUBJ:4(1)
    ENDING AT 4:
     MONITORING [  MODAL  ]
     MONITORING [  NEG  ]
     MONITORING [  V  ]
      NOTICING "IS"
      NOTICING "ARE"
      NOTICING "PAY"

NP/ WAS NEVER PUSHED FOR
   PUSH NP/ TRANS #35 FROM 46:S/Q:4(1) TO 39:S/NP:19(7)
    ENDING AT 4:
     MONITORING [  QADV  ]

NP/ WAS NEVER PUSHED FOR
   PUSH NP/ TRANS #36 FROM 47:VP/HEAD:4(1) TO
        48:VP/NP:19(7)
    ENDING AT 4:
     MONITORING [  PARTICLE  ]
     MONITORING [  V  ]
      NOTICING "PAY"
   JUMP TRANS #37 FROM 49:VP/V:4(1) TO 47:VP/HEAD:4(1)
    ENDING AT 4:
     MONITORING [  NP/ NP/  ]
     MONITORING [  V  ]
      NOTICING "IS"
      NOTICING "ARE"
      NOTICING "PAY"
     MONITORING [  V  ]
      NOTICING "IS"
      NOTICING "ARE"
      NOTICING "PAY"
     MONITORING [  ADV  ]
     MONITORING [  V  ]
   JUMP TRANS #38 FROM 45:S/AUX:4(1) TO 49:VP/V:4(1)
   JUMP TRANS #39 FROM 43:S/NO-SUBJ:4(1) TO 49:VP/V:4(1)
   JUMP TRANS #40 FROM 43:S/NO-SUBJ:4(1) TO 49:VP/V:4(1)
   JUMP TRANS #41 FROM 50:S/THERE:4(1) TO 49:VP/V:4(1)
    ENDING AT 4:
     MONITORING [  THERE  ]
      PROPOSING "THERE"

NP/ WAS NEVER PUSHED FOR
   PUSH NP/ TRANS #42 FROM 51:VP/NP:4(1) TO 52:VP/VP:19(6)
    ENDING AT 4:
     MONITORING [  NP/  ]
   JUMP TRANS #43 FROM 47:VP/HEAD:4(1) TO 51:VP/NP:4(1)

NP/ WAS NEVER PUSHED FOR
   PUSH NP/ TRANS #44 FROM 49:VP/V:4(1) TO 44:VP/V:19(7)
```

```
SELECTED CONFIGS (35 37 39 44 48) FOR EXTENSION
PICKING UP CONFIG 35:TO/:19(7)
   STARTING AT 19:
    MONITORING [  NEG  ]
    MONITORING [  TO  ]
     PROPOSING "TO"
 ALL ARCS TRIED AT THIS CONFIG

PICKING UP CONFIG 37:PP/NP:19(7)
 TRYING POP ARC
  POP TRANS #45 FROM 37:PP/NP:19(7)

PICKING UP CONFIG 39:S/NP:19(7)
   STARTING AT 19:
    MONITORING [  MODAL  ]
    MONITORING [  V  ]
 TRYING POP ARC
  POP TRANS #46 FROM 39:S/NP:19(7)
    EXECUTING PATH (32 31 46)
     BEGINNING AT TRANS 32, CONFIG 42
****
    MADE #3 FROM 4 TO 19:
    S NPU
      NP DET ART THE
         N REGISTRATION
         FEATS NU SG
    WITH FEATURES (NPU)
****
SYN WEIGHT + SEM WT = 10 + 0 = 10

PICKING UP CONFIG 44:VP/V:19(7)
   STARTING AT 19:
    MONITORING [  NP/  ]
     SETTING UP CONFIG 20:NP/:19(7)
    MONITORING [ N ]
     NOTICING "FEE"
    MONITORING [ QDET ]
    MONITORING [ ADJ ]
    MONITORING [ INTEGER ]
    MONITORING [ ART ]
    MONITORING [ QUANT ]
    MONITORING [ PRO ]
    MONITORING [ NPR ]
    MONITORING [ POSS ]
    MONITORING [ V ]
    MONITORING [  V  ]
    MONITORING [  ADV  ]
    MONITORING [  TEST(NOT (CAT V))  ]
     NOTICING "FEE"
 ALL ARCS TRIED AT THIS CONFIG
```

```
PICKING UP CONFIG 48:VP/NP:19(7)
   STARTING AT 19:
    MONITORING [  COMPL/ TO/ COMPL/ NP/  ]
     SETTING UP CONFIG 53:COMPL/:19(7)
    MONITORING [ FOR ]
    MONITORING [ THAT ]
     SETTING UP CONFIG 35:TO/:19(7)
    MONITORING [ TO ]
     SETTING UP CONFIG 53:COMPL/:19(7)
    MONITORING [ FOR ]
    MONITORING [ THAT ]
     SETTING UP CONFIG 20:NP/:19(7)
    MONITORING [ N ]
     NOTICING "FEE"
    MONITORING [ QDET ]
    MONITORING [ ADJ ]
    MONITORING [ INTEGER ]
    MONITORING [ ART ]
    MONITORING [ QUANT ]
    MONITORING [ PRO ]
    MONITORING [ NPR ]
    MONITORING [ POSS ]
    MONITORING [ V ]
    MONITORING [  PARTICLE  ]
 TRYING JUMP VP/VP ARC
  JUMP TRANS #47 FROM 48:VP/NP:19(7) TO 52:VP/VP:19(9)

SELECTED CONFIGS (52) FOR EXTENSION
PICKING UP CONFIG 52:VP/VP:19(9)
   STARTING AT 19:
    MONITORING [  PP/  ]
     SETTING UP CONFIG 30:PP/:19(9)
    MONITORING [ PREP ]
    MONITORING [  PREP  ]
 TRYING JUMP S/VP ARC
  JUMP TRANS #48 FROM 52:VP/VP:19(9) TO 54:S/VP:19(12)

SELECTED CONFIGS (54) FOR EXTENSION
PICKING UP CONFIG 54:S/VP:19(12)
 TRYING JUMP S/S ARC
  JUMP TRANS #49 FROM 54:S/VP:19(12) TO 55:S/S:19(14)

SELECTED CONFIGS (55) FOR EXTENSION
PICKING UP CONFIG 55:S/S:19(14)
 TRYING POP ARC
  POP TRANS #50 FROM 55:S/S:19(14)

PREDICTIONS:
NOTICING (4 FEE 19 23 155 0), SCORE 5
NOTICING (19 WHO 3 4 -180 0), SCORE 10
NOTICING (21 IS 3 4 -39 0), SCORE 10
NOTICING (23 ARE 3 4 -128 0), SCORE 10
```

```
NOTICING (25 PAY 3 4 -146 0), SCORE 5
PROPOSING (TO) STARTING AT 19
PROPOSING (ONLY FOR WHOM WHICH THERE) ENDING AT 4
PROPOSING (V MODAL) FROM 3 TO 4
PROPOSING (MODAL PREP) STARTING AT 3
PROPOSING (PREP MODAL NEG QADV) ENDING AT 4
          {The lengthy summary of monitors  set  by  this
          event is omitted.}

CREATING THEORY 3:
0 WHAT 3   4 THE 6 REGISTRATION 19  23
 WITH SYN SCORE 15

66 MONITORS SET
   21 WORDS
   36 CATEGORIES
   8 CONSTITS
   1 TESTS
9 PROPOSALS MADE
   5 WORDS
   4 CATEGORIES
6 NOTICES MADE
   5 WORDS
   1 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
3 CONSTITS MADE
   0 FOR OTHER THEORIES
   2 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {This took 30.8 seconds.}


          {Now we will process the constituent event  for
          the   theory  just  created.  Because  of  the
          constituent for "the registration" there  are
          now  monitors  in  the  WFST  for a noun phrase
          beginning at position  4,  so  the  appropriate
          transitions are made.}

SYNTAX PROCESSING EVENT FOR THEORY#3 WITH CONSTITUENT #1
  TO GET NEW THEORY#4
0 WHAT 3   4 THE 6 REGISTRATION 19 FEE 23

          {Processing begins exactly where  it  left  off
          when   the   constituent   was   made   --  the
          constituent is semantically evaluated  with
          respect  to this theory so that the constituent
          weight may be altered.  In this case, however,
```

```
          Semantics has been turned off, so there is no
          increment in the score.}
SYN WEIGHT + SEM WT = 15 + 0 = 15

NP/ WAS PUSHED FOR AT CONFIG 34
  PUSH NP/ TRANS #51 FROM 34:FOR/FOR:4(1) TO 56:TO/:23(11)
           {The monitors set by these paths are copied
           from the previous theory, so there is no
           indication here of a new monitor being
           created.}

NP/ WAS PUSHED FOR AT CONFIG 36
  PUSH NP/ TRANS #52 FROM 36:PP/PREP:4(1) TO
 57:PP/NP:23(11)

NP/ WAS PUSHED FOR AT CONFIG 38
  PUSH NP/ TRANS #53 FROM 38:R/NIL:4(1) TO 58:S/NP:23(10)

NP/ WAS PUSHED FOR AT CONFIG 40
  PUSH NP/ TRANS #54 FROM 40:R/WH:4(1) TO 58:S/NP:23(10)

NP/ WAS PUSHED FOR AT CONFIG 41
  PUSH NP/ TRANS #55 FROM 41:S/DCL:4(1) TO 58:S/NP:23(10)

NP/ WAS PUSHED FOR AT CONFIG 43
  PUSH NP/ TRANS #56 FROM 43:S/NO-SUBJ:4(1) TO
 59:VP/V:23(10)

NP/ WAS PUSHED FOR AT CONFIG 46
  PUSH NP/ TRANS #57 FROM 46:S/Q:4(1) TO 58:S/NP:23(10)

NP/ WAS PUSHED FOR AT CONFIG 47
  PUSH NP/ TRANS #58 FROM 47:VP/HEAD:4(1) TO
 60:VP/NP:23(10)

NP/ WAS PUSHED FOR AT CONFIG 51
  PUSH NP/ TRANS #59 FROM 51:VP/NP:4(1) TO 61:VP/VP:23(10)

NP/ WAS PUSHED FOR AT CONFIG 49
  PUSH NP/ TRANS #60 FROM 49:VP/V:4(1) TO 59:VP/V:23(10)

SELECTED CONFIGS (58 57) FOR EXTENSION
PICKING UP CONFIG 58:S/NP:23(10)
 TRYING POP ARC
  POP TRANS #61 FROM 58:S/NP:23(10)
    EXECUTING PATH (32 55 61)
     BEGINNING AT TRANS 55, CONFIG 41
    DOING PUSH ARC WITH #1 FROM 41:S/DCL TO 58:S/NP
    DOING POP ARC FROM 58:S/NP
```

```
****
    MADE #4 FROM 4 TO 23:
    S NPU
      NP DET ART THE
         ADJ NP N REGISTRATION
                  NU SG
         N FEE
         FEATS NU SG
    WITH FEATURES (NPU)
****
SYN WEIGHT + SEM WT = 15 + 0 = 15

PICKING UP CONFIG 57:PP/NP:23(11)
 TRYING POP ARC
  POP TRANS #62 FROM 57:PP/NP:23(11)

PREDICTIONS:
NOTICING (19 WHO 3 4 -180 0), SCORE 10
NOTICING (21 IS 3 4 -39 0), SCORE 10
NOTICING (23 ARE 3 4 -128 0), SCORE 10
NOTICING (25 PAY 3 4 -146 0), SCORE 5
PROPOSING (V MODAL) FROM 3 TO 4
PROPOSING (MODAL PREP) STARTING AT 3
PROPOSING (PREP MODAL NEG QADV) ENDING AT 4
          {Again, the monitor list is omitted for
          considerations of space.}

CREATING THEORY 4:
0 WHAT 3   4 THE 6 REGISTRATION 19 FEE 23
 WITH SYN SCORE 15

33 MONITORS SET
   13 WORDS
   20 CATEGORIES
   0 CONSTITS
   0 TESTS
4 PROPOSALS MADE
   0 WORDS
   4 CATEGORIES
4 NOTICES MADE
   4 WORDS
   0 CONSTITS(THIS THEORY)
   0 CONSTITS(OTHER THEORIES)
0 CONSTITS FOUND
1 CONSTITS MADE
   0 FOR OTHER THEORIES
   12 FOR THIS THEORY
   0 COMPLETE PARSINGS FOUND
          {This event took only 9.7 seconds.}
```

The processing of the final event, that which adds the word "is" to the theory just created, will not be shown.

6.6 SUMMARY

These examples have shown that SPARSER is a useful tool in the automatic recognition of speech. The timing measurements indicate that considerable processing is done when the parser is forced to work in bottom up mode, especially with a large grammar. Of course there is some implementaion overhead involved in doing the timings themselves. If the parseing algorithm were to be carefully recoded in assembly language a speed up of at least a factor of 20 (and perhaps much more) could be achieved. Another way to cut down the time-consuming processing might be to attempt to obtain more semantic guidance. For example, if the semantic hypothesis associated with a theory indicates that a particular noun is likely to be used in a noun phrase modifier (e.g. "tomorrow"), then SPARSER should be able to take advantage of this information by scoring the PUSH NP/ transition from a configuration for state PP/ (i.e. to get something like "by tomorrow") higher than those PUSH NP/ transitions

for other syntactic slots. In fact, the others may not need to be constructed at all. The grammar could also be further tuned to eliminate some spurious predictions and reduce the time spent following erroneous paths.

The following chapter discusses these and other extensions to SPARSER.

Chapter 7

Conclusions and Further Research

7.1 STRENGTHS AND WEAKNESSES OF SPARSER

One of the weak points of the current system is the fact that some context information is not used until a path is complete, resulting in the creation of false paths and predictions which should not have been made. This is partly mitigated by the fact that this avoids a too great dependence on left context and allows the creation of partial paths which may be followed if an earlier word is changed.

It is important, however, to minimize the number of predictions which are made and to make the predictions as accurate as possible. In this regard, it is unfortunate that the current system makes predictions on the left of an island solely on the basis of the first word in the island and makes predictions on the right end from configurations which, if context sensitive tests had been done along the path, would never have been created.

One way to help tighten the predictions would be to take each context free path through an island and walk it in a special mode after the island has been processed but before predictions are communicated to the control component. This mode would set and check registers, assuming that any tests which require unknown left context are true. Only if the path did not fail under this mode of operation would the predictions at either end of it be made. If a really efficient way of handling unknown left context and of storing this information were developed, it could be used in place of the context free pass in the first place, thus eliminating all inconsistent paths.

The problem with storing all possible contexts is that they must be recomputed each time a new step is added to the path. This is relatively easy if the next step is taken to the right of an existing path, since ATN's are more suited to left to right processing, but it becomes extremely complex when a transition is added to the left end of a path (or set of paths) or when a transition joins two sets of paths together. To be absolutely sure that no contexts have been missed, all the paths would have to be walked and their contexts reprocessed and copied in whole or in part (since the new step may be wrong, the old context must be preserved.) Of course this is not the only approach which could be used -- a merging technique like

that of Earley's algorithm (see Section 2.2.1) might be feasible, if the structure of the grammar were also changed to make it less left to right oriented.

Another way to tighten up predictions on the left of an island might be to process the island in an "analysis by synthesis" mode, assuming that a word (or words) which satisfies the left end prediction has been found and setting registers using this assumption. The predictions would not be made if a path blocks as a consequence of this assumption. The problem here is that when a path blocks it is difficult to tell what really caused the problem. The test which ultimately failed could depend on a register whose contents was set as a consequence of a test on a previous register which could have itself been set as a consequence of the original assumption. Since the tests and actions on the arcs of the grammar may be arbitrary functions, all such functions would require modification in order to be able to follow the consequences of a particular action; this is not a burden which should be placed on the writer of the grammar.

Another weakness of SPARSER is that there is no opportunity to share information among islands except that in the special case where they are separated by a one-word gap the islands may co-operate to predict that word. Establishing a global data area associated with each

island where information concerning the nature of the best path through the island (e.g. that the main verb of the sentence has been tentatively found, that part of an auxiliary structure has been found, etc.) would allow compatible paths in subsequent islands to be followed in preference to incompatible ones, especially if context sensitive information were available from registers.

One great strength of the system is its ability to store and merge information in such a way that it does not have to be redone when the context is changed. For example, once an arc has been tried with a particular word match, a transition will be created if the arc may be taken and the arc will be removed from further consideration if it may not be taken. Then, if the configuration should ever be reached with the same word match again (perhaps in a later theory) not only will any relevant transitions be recognized without having to go through the work of re-creating them , but also no arcs which had previously failed will be retried.

Another feature of SPARSER is the fact that it was designed and implemented with many unsolved problems and unavailable data in mind, and therefore many "holes" have been left on which to "hook" further developments. For example, although prosodic verification of constituents is not yet available, the scoring mechanism for constituents

is structured in such a way that it would be easy to include the results of verification by prosodics (or any other component). The original implementation of SPARSER used a depth first search but was implemented in such a way that the change to modified breadth first was quite simple. This foresight has paid off in a flexible system which has shown that it can be readily experimented with in order to explore many still unsolved problems concerning the nature and use of syntactic information in understanding.

7.2 PROSODICS

A tremendous amount of information in speech is conveyed by prosodic features: stress, intonation, duration, loudness, pauses, pitch. For example, if John mumbles to Bill, "The mailman left something for you," Bill may reply either "What?" with much energy and a sharply rising intonation or "What?" with a flat or falling intonation. In the first case John is very likely to shout "I said, 'The mailman left something for you'", interpreting "What?" to mean "What did you say?" whereas in the second case he is likely to say something like "A package from your mother," interpreting "What?" to mean "What is it?"

The case where virtually all the disambiguation information is carried by prosodic features is fairly rare in English (though not in languages such as Chinese). It is more common for important information in speech to be encoded redundantly, by a combination of syntactic, semantic, pragmatic, prosodic, and phonological cues. To ignore prosodics, however, is to ignore a source of information which has been shown repeatedly to be an extremely important factor in human understanding [22, 30, 48, 49].

Consider the following examples of sentences and sentence fragments which illustrate some of the ways prosodies are used:

1. I stepped on the man with black shoes. (Who was wearing the shoes?)

2a. The new gnu knew news.

2b. The gnu knew new gnus.

3. I'm going to move on Thursday. (stress on "move" implies moving to a new house; stress on "on" implies going to visit a new place.)

4a. Can you swim to Daddy?

4b. Can you swim too, Daddy?

5a. In the process of computing, the average values will be used.

5b. In the process of computing the average, values will

be used.

5c. The process of computing the average values will be used.

5d. In the process, computing the average values will be used.

6a. ... two-fifty for ...

6b. ... two-fifty-four ...

Prosodic verification could help a lot in rejecting semantically correct, syntactically consistent phrases which are nonetheless wrong. If the constituent "speech understanding" were identified and relied upon, it might be very difficult to produce a correct analysis of the utterance: "Because of peculiarities in his speech, understanding Joe is not easy."

Besides indicating syntactic boundaries and/or providing intonation contours for certain constituents, prosodic features can be used to mark emphasis, introduce new topics, convey information about the speaker's internal mental and emotional state (e.g. whether he is teasing or serious), and probably more. It is particularly interesting to note that some well known phenomena such as "pronouns are almost never stressed" and "in discourse when a new topic word is mentioned it is almost always stressed" have very natural explainations in light of what we know about acoustic processing. Stressed

words are generally easier to identify because there is less acoustic ambiguity, but unstressed words may differ greatly from their ideal pronunciation and hence are harder to reliably identify. Pronouns refer to antecedents which are presumably known to the listener, so he can anticipate them or at least verify them easily, hence they need not have good acoustic characteristics. A new topic may not have been anticipated, so the listener will have to depend heavily on identifying the word from acoustic information alone and the speaker can provide this extra reliable information by stressing the word.

Unfortunately, not a great deal is known about either the acoustic correlates of prosodic features or the ways in which they are used (see [47, 48, 58]). Many of the rules which have been developed thus far (such as Jon Allen's work with fundamental frequency patterns in auxiliaries [2]) are speaker dependent and are sufficient for conveying information but are not necessary (as shown in an auxiliary example by Yeager and O'Malley [98]). This makes them difficult to use in the analysis mode. Although a good start has been made in exploring prosodies, much more work remains to be done before prosodic information can be reliably used by speech understanding systems.

Considerable work has been done by Wayne Lea [44-46] to develop an algorithm to detect certain types of syntactic boundaries based on fluctuations in the fundamental frequency contour. Experimentation with his algorithm by Bates and Wolf [8] has shown that while many boundaries are detected, two facts stand out which make the process difficult to use in speech analysis. First, the boundaries which are "detected" are not "located", that is, the point in the utterance where the Lea-boundary is found may be anywhere from the first stressed vowel following the actual boundary to the first stressed vowel before it. In addition, a rather high percentage (23%) of the detections are spurious. Some types of boundaries are reliably detected by this algorithm, however. It found 23 out of 24 prepositional phrase boundaries in a 16 sentence test. Perhaps by reinterpreting the results of the experiment or by varying the thresholds used in the detection algorithm it may be possible to reduce the percentage of false drops and to make the remaining detections more reliable so that the information can be useful in a speech understanding system. Continued investigation of these possibilities is currently under way at BBN.

SPARSER could use prosodic information in several ways. Verification of constituents would be a great help, but local prosodic information could be used even earlier in the parsing process. For example, if major constituent boundaries could be accurately determined, then instead of both POPing a constituent and continuing it in parallel, as is done now, one alternative could be chosen instead of the other on the basis of prosodic information. If, as is more likely, some major boundaries could be reliably detected, then it would be easy to revise SPARSER to begin processing at such places even within an island at states which can begin constituents. This would again reduce the number of partial paths created when parsing an island.

7.3 EXTENSIONS AND FURTHER RESEARCH

7.3.1 Ungrammatical input

One of the obvious extensions to a basic speech understanding system is to relax the restrictions on the input to the system. Syntactically, this can mean removing the requirement that the initial utterance be grammatical. Since people frequently speak ungrammatically in informal discourse, this is a natural step to want to take.

In order to extend SPARSER to handle such input, several approaches are possible. Certain types of errors may be called errors of style (and may not be called errors at all by some people) such as the use of "ain't" and the occurrence of a preposition at the end of a sentence. These regularities may simply be declared grammatical by modifying the grammar to accept them. Many speech errors have been shown to follow regular patterns (see Fromkin [27]) and hence may be amenable to this approach.

Other common errors violate specific tests which appear on the arcs of the grammar, for example, to prohibit double negatives and to check for number agreement between subject and verb or between determiner and noun (e.g. *"There is some very severe restrictions on this rule."). In this case, rather than removing the tests from the grammar it would be more suitable to modify them so that if they failed the arc could still be taken, though with a much reduced weight or with an indication in some register that an error has occurred. One way to implement this would be to have all tests return a number as their value indicating how well they succeeded on some scale from "perfectly" to "not at all".

Not all arc tests are of this relaxable nature, however, since certain types of errors are so rare, if they occur at all, that they may be judged unacceptable. Examples of such tests are the case checks for pronouns (e.g. *"I gave it to he") and the requirement that a verb modifying a noun must be in either the present or past participle form (e.g. "the singing brook" vs. *"the sing brook").

These methods would not allow all types of grammatical errors to be handled (in particular it ignores the problem of constituent ordering errors such as "Throw Mama from the train a kiss"), but would handle many of the most common syntactic errors.

7.3.2 An experiment

Keeping in mind the caveat of Section 1.3, that SPARSER is not intended to be a model of human syntactic analysis, it is nonetheless reasonable to ask whether there are any similarities which may be seen. The following experiment is suggested with the hypothesis that it will indicate that people do considerable processing at the end of syntactic constituents in a way similar to some register setting and testing actions and semantic (or other) verification.

The experiment is this: a subject is seated in front of a switch which he is asked to press whenever he is sure that he is hearing an anomalous sentence. He is then presented with a sequence of recorded utterances, some of which are incorrect, e.g.

*The cat and dog which lives next door are friendly.

*I saw a red big barn on the farm.

I hypothesize that the subject will indicate the presence of an error at a point shortly after the end of the constituent in which the error occurred more often than shortly after the earliest possible place where the error could be detected.

7.4 CONCLUSION

In conclusion, it is obvious that there is much work yet to be done in the problem of speech understanding, but it is hoped that the system presented here has not only advanced our current understanding of the role of syntactic knowledge in comprehension but will continue to be useful for further exploration of the field.

APPENDIX I

MINIGRAMMAR

This appendix contains a listing (slightly edited for clarity) of the grammar called MINIGRAMMAR which was discussed in Chapter Three (illustrated in Figure 3.3) and which was used in some of the examples in Chapter Six.

```
(NP/
   (CAT ART (T T)
       5
       (SETR ART (BUILDQ ((ART *))))
       (TO NP/ART))
   (JUMP NP/ART (T T)
       4))

(NP/ADJ
   (CAT N (T T)
       5
       (SETR N *)
       (SETR NU (GETF NUMBER))
       (TO NP/N))
   (CAT N (T T)
       2
       (ADDL ADJS (BUILDQ (ADJ (NP (N *)
                                   (NU  )))
                          (GETF NUMBER)))
       (TO NP/ADJ)))

(NP/ART
   (CAT QUANT (T T)
       4
       (SETR QUANT (BUILDQ ((QUANT *))))
       (TO NP/QUANT))
   (JUMP NP/QUANT (T T)
       5))
```

```
(NP/QUANT
   (CAT ADJ (T T)
        4
        (ADDR ADJS (BUILDQ (@ (ADJ)
                              (*))
                           FEATURES))
        (TO NP/QUANT))
   (JUMP NP/ADJ (T T)
        4))

(NP/N
   (PUSH PP/ ((PPSTART)
              T T)
        4
        (ADDL NMODS *)
        (TO NP/N))
   (POP (BUILDQ (@ (NP)
                   + + + ((N +))
                   ((NU +))
                   +)
                ART QUANT ADJS N NU NMODS)
        (T (DETAGREE))
        5))

(PP/
   (CAT PREP (T T)
        5
        (SETR PREP *)
        (TO PP/PREP)))

(PP/PREP
   (PUSH NP/ ((NPSTART)
              T T)
        5
        (SETR NP *)
        (TO PP/NP)))

(PP/NP
   (POP (BUILDQ (PP (PREP +)
                    +)
                PREP NP)
        (T T)
        5))
```

APPENDIX II

SPEECHGRAMMAR

This appendix contains a complete listing of the SPEECHGRAMMAR discussed in Chapter Four and used in the examples in Chapter Six. Due to considerations of space, complete listings of the functions which are called on the arcs of the grammar (for example, tests and some structure building operations) were not included, but the names of the functions should be sufficiently indicative of the purpose to be comprehensible. Forms beginning with a * are comments.

```
(FILECREATED "22-JUL-75 13:35:40"
<BATES>SPEECHGRAMMAR.;2 37501  )
  (LISPXPRINT (QUOTE SPEECHGRAMMARCOMS)
              T T)
  (RPAQQ SPEECHGRAMMARCOMS
         ((G: SPEECHGRAMMAR)
          (V: (DIGITS TEENS TENS CLASSES /MONTH/
                      /WEEKDAY/))))

(LISPXPRINT (QUOTE (G: SPEECHGRAMMAR)) T)

(RPAQQ SPEECHGRAMMAR ($/ $/$ $/AND $/CENTS $/DOLLARS
$/NUM COMPL/ COMPL/NTYPE COMPL/S DATE/ DATE/DATE
DATE/DAY DATE/MO DATE/NUM DATE/OF FOR/FOR FOR/TO
NP/ NP/ART NP/D NP/DET NP/HEAD NP/NP NP/ONLY NP/ORD
NP/QUANT NP/R NPR/ NPR/CITY NPR/FIRST NPR/NPR NUM/
NUM/1 NUM/NUM NUM/STR NUMBER/ NUMBER/1 NUMBER/1A
NUMBER/A NUMBER/H NUMBER/H2 NUMBER/K NUMBER/NUMBER
NUMBER/OH NUMBER/OH2 NUMBER/OHD NUMBER/P NUMBER/PO
```

```
NUMBER/T NUMBER/T2 NUMBER/TA NUMBER/TH ORD/ ORD/NUM
ORD/ORD PP/ PP/NP PP/PREP QUANT/ QUANT/QUANT QUANT/UNIT
R/ R/NIL R/PREP R/WH R/WHOSE S/ S/AUX S/DCL S/HOW
S/IF S/IFTHEN S/IMP S/NO-SUBJ S/NP S/Q S/S S/THEN
S/THERE S/VP TO/ VP/HEAD VP/NP VP/V VP/VP))

(DEFINEQ

($/
   (PUSH NUMBER/ ((NUMBERSTART)
                  T
                  (NOR (GETF DIGITS)
                       (GETF AMBIG)
                       (GETF OH)))
         3
         (SETR NUM (NUMBOF *))
         (TO $/NUM))
   (PUSH NUMBER/ ((NUMBERSTART)
                  T
                  (OR (FLOATP (NUMBOF *))
                      (GETF AMBIG)))
         2
         (COND ((GETF AMBIG)
                (SETR NUM (FQUOTIENT (NUMBOF *)
                                     100)))
               (T (SETR NUM (NUMBOF *))))
         (TO $/$)))

($/$
   (POP (BUILDQ (N  )
                (PACK (CONS (QUOTE "$")
                            (UNPACK (GETR NUM)))))
        (T T)
        5))

($/AND
   (PUSH NUM/ (T T (ILESSP (NUMBOF *)
                           100))
        5
        (SETR NUM (IPLUS (GETR NUM)
                         (FQUOTIENT (NUMBOF *)
                                    100)))
        (TO $/CENTS)))

($/CENTS
   (WRD CENT ((EQ (GETF NUMBER)
                  (QUOTE PL))
              T)
        5
        (TO $/$)))

($/DOLLARS
```

```
   (JUMP $/$ (T T)
       2)
   (WRD AND (T T)
       3
       (TO $/AND)))

($/NUM
   (WRD DOLLAR ((EQ (GETF NUMBER)
                    (QUOTE PL))
                T)
       5
       (TO $/DOLLARS)))

(COMPL/
   (WRD FOR (T T)
       3
       (SETRQ TYPE FORCOMP)
       (TO FOR/FOR))
   (WRD THAT (T T)
       3
       (SETRQ THAT T)
       (TO COMPL/NTYPE)))

(COMPL/NTYPE
   (PUSH S/ ((SSTART)
             T
             (NEQ (STYPE)
                  (QUOTE NPU)))
       5
       (COND
        ((AND (NULLR THAT)
              (EQ (QUOTE THAT)
                  (NP.DET (S.NP (UNHASH *)))))
         (ABORT)))
       (SETR S (CONS (QUOTE S)
                     (CONS (QUOTE THATCOMP)
                           (CDDR (UNHASH *)))))
       (TO COMPL/S)))

(COMPL/S
   (POP (GETR S)
       (T T)
       5))

(DATE/
   (WRD /WEEKDAY/ (T T)
       5
       (SETR DAY *)
       (TO DATE/DAY))
```

```
   (WRD /WEEKDAY/ (T T)
        5
        (SETR DAY *)
        (TO DATE/DATE))
   (PUSH NUM/ ((NUMSTART)
               T
               (ILESSP (NUMBOF *)
                       32))
        2
        (SETR NUM *)
        (TO DATE/DAY))
   (JUMP DATE/DAY (T T)
        4))

(DATE/DATE
   (POP (BUILDQ (@ (DATE)
                   )
                (COND ((GETR DAY)
                       (BUILDQ ((DAY +))
                               DAY)))
                (COND ((GETR NUM)
                       (BUILDQ ((NUM +))
                               NUM)))
                (COND ((GETR MONTH)
                       (BUILDQ ((MONTH +))
                               MONTH)))
                (COND ((GETR YEAR)
                       (BUILDQ ((YEAR +))
                               YEAR))))
        (T (OR (GETR DAY)
               (GETR MONTH)
               (GETR YEAR)))
        5
        (* SHOULD TEST NUMBER OF DAYS IN MONTH TO AVOID
          FEB 30 OR SEPT 31)))

(DATE/DAY
   (WRD /MONTH/ (T (NULLR MONTH))
        5
        (SETR MONTH *)
        (TO DATE/MO))
   (WRD THE (T (NULLR NUM))
        3
        (SETRQ THE T)
        (TO DATE/MO))
   (JUMP DATE/MO (T (NOR (GETR DAY)
                         (GETR NUM)))
        3))
```

```
(DATE/MO
   (PUSH ORD/ ((ORDSTART)
               (NULLR NUM)
               (AND (NUMBERP (NUMBOF *))
                    (ILESSP (NUMBOF *)
                            32)))
       2
       (SETR NUM *)
       (TO DATE/NUM))
   (JUMP DATE/NUM (T (OR (AND (GETR MONTH)
                              (NULLR DAY))
                         (AND (NULLR MONTH)
                              (NULLR DAY)
                              (NULLR NUM)
                              (NULLR THE))))
       4)
   (PUSH NUMBER/ ((NUMBERSTART)
                  (NULLR NUM)
                  (ILESSP (NUMBOF *)
                          32))
       2
       (SETR NUM *)
       (TO DATE/NUM)))

(DATE/NUM
   (PUSH NUMBER/ ((NUMBERSTART)
                  T
                  (YEAR? (UNHASH *)))
       4
       (SETR YEAR *)
       (TO DATE/DATE))
LLR THE))   (JUMP DATE/DATE (T
       4)
   (WRD OF (T (AND (NULLR MONTH)
                   (GETR THE)))
       5
       (SETR THE)
       (TO DATE/OF)))

(DATE/OF
   (WRD /MONTH/ (T T)
       5
       (SETR MONTH *)
       (TO DATE/NUM)))
```

```
(FOR/FOR
   (PUSH NP/ ((NPSTART)
              T
              (NOT (ROLE SUBJ)))
         5
         (* "IT IS TIME FOR ME TO LEAVE")
         (SETR SUBJ *)
         (TO TO/)))

(FOR/TO
   (CAT V ((CHECKF V UNTENSED)
           T)
         5
         (SETR V *)
         (SETRQ VOICE ACTIVE)
         (TO VP/V)))

(NP/
   (WRD ONLY (T T)
         5
         (* "ONLY A FOOL WOULD X" , "ONLY THREE FLIGHTS"
           "ONLY THE FLIGHTS WHICH X")
         (ADDL PREMODS (BUILDQ (PREDET *)))
         (TO NP/ONLY))
   (CAT QDET (T T)
         5
         (* "WHOSE CAR IS THAT?" "WHICH FIVE TRIPS"
           "HOWMANY (TRIPS) WERE THERE")
         (SETR ART *)
         (ADDL LIFTLIST (QUOTE QDET))
         (SETR QDET T)
         (TO NP/ORD))
   (PUSH DATE/ ((DATESTART)
                T T)
         1
         (SETR DATE *)
         (TO NP/NP))
   (TST DATEMOD ((DATEMOD)
                 T)
         4
         (SETR PREMODS (BUILDQ ((ADJ *))))
         (SETRQ NPNUAGREE T)
         (TO NP/D))
   (JUMP NP/ONLY ((NOR (CAT QDET)
                       (WRD ONLY))
                  T)
         4))
```

```
(NP/ART
   (PUSH $/ ((START)
             T T)
        1
        (SETR DET (DETBUILD))
        (SETR HEAD *)
        (SETRQ NU PL)
        (TO NP/HEAD))
   (PUSH NUMBER/ ((NUMBERSTART)
                  T T)
        1
        (SETR HFAD (BUILDQ (NUMBER *)))
        (SETRQ NU PL)
        (SETR DET (DETBUILD))
        (SETR LIFTLIST FEATURES)
        (TO NP/HEAD))
   (WRD ONLY (T (ART THE))
        5
        (* "THE ONLY ONE")
        (ADDL PREMODS (BUILDQ (POSTDET *)))
        (TO NP/ORD))
   (CAT ORD (T (GETR ART))
        5
        (* "THE FIRST SAMPLE" , "THE NEXT OF THE ROCKS")
        (SETR POSTART (BUILDQ ((ORD *))))
        (TO NP/ORD))
   (JUMP NP/ORD (T T)
        4))

(NP/D
   (WRD /MONTH/ (T T)
        3
        (SETR HEAD (BUILDQ (DATE (MONTH *))))
        (SETRQ NU SG)
        (TO NP/NP))
   (PUSH NUMBER/ ((NUMBERSTART)
                  T
                  (YEAR? (UNHASH *)))
        3
        (SETR HEAD (BUILDQ (DATE (YEAR *))))
        (SETRQ NU SG)
        (TO NP/NP)))

(NP/DET
   (CAT N (T T)
        5
        (SETR HEAD (BUILDQ (N *)))
        (SETR NU (GETF NUMBER))
        (TO NP/HEAD))
```

```
(PUSH NPR/ ((NPRSTART)
            T T)
     3
     (SETR HEAD *)
     (SETRQ NU SG)
     (TO NP/HEAD))
(CAT ADJ (T T)
     4
     (COND ((NULLR ADADJ)
            (* ADADJ ARE LIKE ADVERBS ONLY THEY
               MODIFY ADJECTIVES RATHER THAN VERBS.
               WE HAVE NONE IN THE CURRENT
               VOCABULARY. WHEN WE DO, IT WILL BE
               NECESSARY TO PUT A
               (CAT ADADJ (T T)
                    (SETR ADADJ *)
                    (TO NP/DET))
               IN NP/QUANT)
            (ADDL PREMODS (BUILDQ (@ (ADJ)

                                     (*))
                                  FEATURES)))
           ((ADJAGREE (GETR ADADJ)
                      *)
            (ADDL PREMODS
                  (BUILDQ (ADJP (ADV +)
                                (ADJ *))
                          ADADJ))
            (SETR ADADJ))
           (T (* DO SOMETHING LIKE LOWERING WEIGHT
                 ON PATH)))
     (TO NP/DET))
(CAT N (T (NULLR ADADJ))
     3
     (ADDL PREMODS
           (BUILDQ (ADJ (NP (N *)
                            (NU  )))
                   (GETF NUMBER)))
     (TO NP/DET))
(PUSH NPR/ ((NPRSTART)
            T T)
     2
     (ADDL PREMODS (BUILDQ (ADJ (NP *)
                                (NU SG))))
     (TO NP/DET))
```

```
   (CAT V ((OR (GETF PRESPART)
               (GETF PASTPART))
           T)
        3
        (* "A SINGING BIRD" , "AN ELEVATED PLATFORM")
        (ADDL NMODS (BUILDQ (ADJ (PARTICIPLE
                                   *
                                   (TNS  )))
                                 (COND ((GETF PRESPART)
                                        (QUOTE PRES))
                                       (T (QUOTE PAST)))))
        (TO NP/DET))

   (CAT ADV (T T)
        2
        (* SOME ADVS CANNOT APPEAR HERE, E.G. THERE.
          THEY SHOULD BE SCREENED OUT)
        (* THE VERY OLD CAR, THE UNUSUALLY INEXPENSIVE
          TRIP)
        (SETR ADV *)
        (TO NP/ADV))
   (JUMP NP/HEAD (T (AND (NULLR PREMODS)
                         (ELLIP? (GETR DET))))
        3
        (* "I NEED SEVEN" , "ALL ARE GONE")
        (SETRQ HEAD (PRO ONE))
        (SETR NU (GETDETNUM (GETR DET)))))

(NP/HEAD
   (PUSH R/ ((RELSTART)
             (OR (DET.ART (GETR DET))
                 (WRD PL NU)
                 (GETF N MASS)
                 (RELAGREE))
             T)
        3
        (* "THE TRIP WHICH JOHN TOOK" ,
          "GIRLS TO WHOM I WROTE" ,
          "MILK WHICH IS SOUR")
        (ADDL NMODS *)
        (TO NP/R))
   (PUSH PP/ ((PPSTART)
              T T)
        3
        (* "THE BOOKS ON THE SHELF")
        (ADDL NMODS *)
        (TO NP/HEAD))
```

```
   (PUSH R/NIL ((RRELSTART)
                (AND (NULLR PPFLAG)
                     (NULLR QDET)
                     (RELAGREE))
                T)
       3
       (* "THE TRIP JOHN TOOK")
       (ADDL NMODS *)
       (TO NP/R))
   (POP (NPBUILD)
       (T (NPNUCHEK))
       5))

(NP/NP
   (POP (NPBUILD)
       (T T)
       5))

(NP/ONLY
   (JUMP NP/ART ((NOR (CAT (ART PRO NEG QUANT POSS))
                      (WRD (WHOSE WHO HOW WHAT WHETHER
                                  IF WHICH)))
                 T)
       4)
   (CAT ART (T T)
       3
       (SETR ART (BUILDQ ((ART *))))
       (TO NP/ART))
   (CAT QUANT (T T)
       3
       (* "ALL THE MONEY" "ENOUGH MONEY"
         "HOWMUCH MONEY")
       (SETR QUANT (BUILDQ ((QUANT *))))
       (TO NP/ART))
   (CAT POSS (T T)
       3
       (* "HIS TRAVEL PLANS")
       (ADDL NMODS *)
       (TO NP/ART))
   (CAT PRO (T T)
       3
       (SETR HEAD (BUILDQ (PRO *)))
       (SETR NU (GETF NUMBER))
       (SETR DET (DETBUILD))
       (COND ((GETF ROLE)
              (SETR ROLE (GETF ROLE))))
       (TO NP/NP)))
```

```
(NP/ORD
   (PUSH QUANT/ ((QUANTSTART)
                 T T)
       2
       (* AN ORDINAL INDICATOR HAS BEEN ANALYZED
         (OR IT WAS OMITTED)
         AND AN OPTIONAL QUANTIFIER MAY FOLLOW
         "THE (FIRST) FIVE MEN")
       (SETR POSTART (BUILDQ ((@ (POSTART)
                                 +
                                 (*)))
                              POSTART))
       (TO NP/QUANT))
   (JUMP NP/QUANT (T T)
       5))

(NP/QUANT
   (JUMP NP/DET (T T)
       5
       (SETR DET (DETBUILD))))

(NP/R
   (JUMP NP/NP (T T)
       5))

(NPR/
   (CAT FIRSTNAME (T T)
       3
       (SETR NAME T)
       (SETR FIRST *)
       (TO NPR/FIRST))
   (CAT CITY (T T)
       3
       (SETR LOCATION T)
       (SETR CITY *)
       (TO NPR/CITY))
   (JUMP NPR/CITY (T T)
       2
       (SETR LOCATION T))
   (CAT NPR (T T)
       2
       (SETR NPR *)
       (TO NPR/NPR)))

(NPR/CITY
   (POP (BUILDQ (NPR (LOCATION (CITY +)))
                CITY)
       (T (GETR CITY)
         3))
```

```
   (CAT STATE (T T)
        3
        (SETR STATE *)
        (TO NPR/NPR))
   (CAT COUNTRY (T T)
        3
        (SETR COUNTRY *)
        (TO NPR/NPR)))

(NPR/FIRST
   (CAT LASTNAME . T)
        4
        (SETR LAST *)
        (TO NPR/NPR))
   (JUMP NPR/NPR (T T)
        2))

(NPR/NPR
   (POP (NPRBUILD)
        (T T)
        5))

(NUM/
   (CAT INTEGER ((OR (MEMB (NUMBOF *)
                           DIGITS)
                     (MEMB (NUMBOF *)
                           TEENS))
                 T)
        4
        (SETR NUM (NUMBOF *))
        (TO NUM/NUM))
   (CAT INTEGER ((MEMB (NUMBOF *)
                       TENS)
                 T)
        4
        (SETR NUM (NUMBOF *))
        (TO NUM/1)))

(NUM/1
   (CAT INTEGER ((MEMB (NUMBOF *)
                       DIGITS)
                 T)
        4
        (SETR NUM (IPLUS (NUMBOF *)
                         (GETR NUM)))
        (TO NUM/NUM))
   (JUMP NUM/NUM (T T)
        3))
```

```
(NUM/NUM
   (POP (GETR NUM)
        (T T)
        5))

(NUM/STR
   (POP (PACK (GETR NUMSTR))
        (T T)
        3
        (SELECTQ (LENGTH (GETR NUMSTR))
                 (4 (SETRQ LIFTLIST (TRIPNO DIGITS)))
                 ((5 6)
                  (SETRQ LIFTLIST (JOBNO DIGITS)))
                 (ABORT)))
   (CAT INTEGER ((OR (MEMB (NUMBOF *)
                           DIGITS)
                     (EQ (NUMBOF *)
                         0))
                 T)
        4
        (* STRINGS OF DIGITS LIKE "ONE OH FIVE ONE OH")
        (ADDR NUMSTR (NUMBOF *))
        (TO NUM/STR)))

(NUMBER/
   (PUSH NUM/ ((NUMSTART)
               T T)
        4
        (SETR TOTAL (NUMBOF *))
        (TO NUMBER/1))
   (PUSH NUM/STR ((NUMSTRSTART)
                  T T)
        3
        (SETR TOTAL (NUMBOF *))
        (SETR LIFTLIST FEATURES)
        (TO NUMBER/NUMBER))
   (WRD (ZERO NO) (T T)
        5
        (SETRQ TOTAL 0)
        (TO NUMBER/NUMBER))
   (WRD A (T T)
        3
        (TO NUMBER/A))
   (WRD POINT (T T)
        5
        (TO NUMBER/PO)))
```

```
(NUMBER/1
   (PUSH NUM/ ((NUMSTART)
               T
               (NOT (MEMB (NUMBOF *)
                          DIGITS)))
        3
        (* LOOKING FOR 250, 1292, FOR EXAMPLE)
        (SETR TOTAL (IPLUS (ITIMES 100 (GETR TOTAL))
                           (NUMBOF *)))
        (TO NUMBER/OHD))

   (WRD OH (T T)
        3
        (* LOOKING FOR 109, 1207, FOR EXAMPLE)
        (TO NUMBER/OH))
   (WRD HUNDRED (T (NOR (EQ (GETR NUM)
                            10)
                        (MEMB (GETR NUM)
                              TENS)))
        5
        (SETR TOTAL (ITIMES 100 (GETR TOTAL)))
        (TO NUMBER/H))
   (WRD THOUSAND (T T)
        5
        (SETR TOTAL (ITIMES 1000 (GETR TOTAL)))
        (TO NUMBER/T))
   (WRD K (T T)
        5
        (SETR TOTAL (ITIMES 1000 (GETR TOTAL)))
        (TO NUMBER/NUMBER))
   (WRD POINT (T T)
        4
        (SETR TOTAL (ITIMES 1000 (GETR TOTAL)))
        (TO NUMBER/P))
   (WRD AND (T T)
        3
        (SETR TOTAL (ITIMES 1000 (GETR TOTAL)))
        (TO NUMBER/1A))
   (JUMP NUMBER/NUMBER (T T)
        2))

(NUMBER/1A
   (WRD (A ONE) (T T)
        2
        (SETRQ A1 A1)
        (TO NUMBER/1A))
   (WRD QUARTER (T (GETR A1))
        2
        (SETR TOTAL (IPLUS (GETR TOTAL)
                           250))
        (TO NUMBER/K))
   (WRD HALF (T (GETR A1))
```

```
        2
        (SETR TOTAL (IPLUS (GETR TOTAL)
                           500))
        (TO NUMBER/K)))

(NUMBER/A
   (WRD HUNDRED (T T)
        2
        (SETRQ TOTAL 100)
        (TO NUMBER/H))
   (WRD THOUSAND (T T)
        2
        (SETRQ TOTAL 1000)
        (TO NUMBER/T)))

(NUMBER/H
   (PUSH NUM/ ((NUMSTART)
               T T)
        3
        (SETR TOTAL (IPLUS (GETR TOTAL)
                           (NUMBOF *)))
        (SETR AND NIL)
        (TO NUMBER/H2))
   (JUMP NUMBER/NUMBER (T (NULLR AND))
        2)
   (WRD AND (T (NULLR AND))
        3
        (SETRQ AND AND)
        (TO NUMBER/H))
   (WRD THOUSAND (T (NULLR AND))
        4
        (SETR TOTAL (ITIMES 1000 (GETR TOTAL)))
        (TO NUMBER/T))
   (WRD K (T T)
        4
        (SETR TOTAL (ITIMES (GETR TOTAL)
                            1000))
        (TO NUMBER/NUMBER)))

(NUMBER/H2
   (WRD K (T T)
        3
        (SETR TOTAL (ITIMES (GETR TOTAL)
                            1000))
        (TO NUMBER/NUMBER))
   (WRD THOUSAND (T T)
        3
        (SETR TOTAL (ITIMES 1000 (GETR TOTAL)))
        (TO NUMBER/T))
```

```
   (JUMP NUMBER/NUMBER (T T)
       2))

(NUMBER/K
   (WRD (K THOUSAND) (T T)
       5
       (TO NUMBER/NUMBER)))

(NUMBER/NUMBER
   (POP (GETR TOTAL)
       (T T)
       5
       (COND ((GETR DIGITS)
              (ADDL LIFTLIST (QUOTE DIGITS))))))

(NUMBER/OH
   (CAT INTEGER ((MEMB (NUMBOF *)
                       DIGITS)
                 T)
       5
       (SETR TOTAL (IPLUS (ITIMES 100 (GETR TOTAL))
                          (NUMBOF *)))
       (SETR DIGITS T)
       (ADDL LIFTLIST (QUOTE OH))
       (TO NUMBER/OHD)))

(NUMBER/OH2
   (CAT INTEGER ((MEMB (NUMBOF *)
                       DIGITS)
                 T)
       5
       (SETR TOTAL (FPLUS (GETR TOTAL)
                          (FQUOTIENT (NUMBOF *)
                                     100)))
       (TO NUMBER/NUMBER)))

(NUMBER/OHD
   (PUSH NUM/ ((NUMSTART)
               T
               (NOT (MEMB (NUMBOF *)
                          DIGITS)))
       3
       (* LOOKING FOR 1207.79, FOR EXAMPLE)
       (SETR TOTAL (FPLUS (GETR TOTAL)
                          (FQUOTIENT (NUMBOF *)
                                     100)))
       (SETR DIGITS NIL)
       (TO NUMBER/NUMBER))
```

```
   (WRD OH (T T)
       3
       (* LOOKING FOR 1207.09, FOR EXAMPLE)
       (TO NUMBER/OH2))
   (JUMP NUMBER/NUMBER (T T)
       2
       (ADDR LIFTLIST (QUOTE AMBIG))))

(NUMBER/P
   (PUSH NUM/ ((NUMSTART)
               T T)
       5
       (COND ((MEMB (NUMBOF *)
                    DIGITS)
              (SETR TOTAL (IPLUS (GETR TOTAL)
                                 (ITIMES 100
                                         (NUMBOF
                                           *)))))
             (T (ABORT)))
       (TO NUMBER/K)))

(NUMBER/PO
   (PUSH NUM/ ((NUMSTART)
               T T)
       5
       (COND ((MEMB (NUMBOF *)
                    DIGITS)
              (SETR TOTAL (ITIMES (NUMBOF *)
                                  100)))
             (T (ABORT)))
       (TO NUMBER/K)))

(NUMBER/T
   (PUSH NUM/ ((NUMSTART)
               T T)
       2
       (COND ((IGREATERP (NUMBOF *)
                         9)
              (SETR TOTAL (IPLUS (GETR TOTAL)
                                 (NUMBOF *))))
             (T (ABORT)))
       (TO NUMBER/NUMBER))
   (PUSH NUM/ ((NUMSTART)
               T T)
       2
       (COND ((ILESSP (NUMBOF *)
                      10)
              (SETR NUM (NUMBOF *)))
             (T (ABORT)))
       (TO NUMBER/TH))
```

```
   (WRD AND (T T)
        3
        (TO NUMBER/TA)))

(NUMBER/T2
   (PUSH NUM/ ((NUMSTART)
               T T)
        3
        (SETR TOTAL (IPLUS (GETR TOTAL)
                           (NUMBOF *)))
        (TO NUMBER/NUMBER))
   (WRD AND (T (NULLR AND))
        3
        (SETRQ AND AND)
        (TO NUMBER/T2))
   (JUMP NUMBER/NUMBER (T (NULLR AND))
        2))

(NUMBER/TA
   (PUSH NUM/ ((NUMSTART)
               T T)
        5
        (SETR TOTAL (IPLUS (GETR TOTAL)
                           (NUMBOF *)))
        (TO NUMBER/NUMBER)))

(NUMBER/TH
   (WRD HUNDRED (T T)
        3
        (SETR TOTAL (IPLUS (GETR TOTAL)
                           (ITIMES 100 NUM)))
        (TO NUMBER/T2))
   (JUMP NUMBER/NUMBER (T T)
        2
        (SETR TOTAL (IPLUS (GETR TOTAL)
                           NUM))))

(ORD/
   (CAT ORD (T T)
        5
        (SETR ORD *)
        (TO ORD/ORD))
   (PUSH NUMBER/ ((NUMBERSTART)
                  T
                  (AND (IGREATERP * 19)
                       (EQ 0 (IREMAINDER * 10))))
        5
        (* "THIRTY FIFTH")
        (SETR ORD *)
        (TO ORD/NUM)))
```

```
(ORD/NUM
   (CAT ORD ((ILESSP * 10)
             T)
        5
        (SETR ORD (IPLUS * (GETR ORD)))
        (TO ORD/ORD)))

(ORD/ORD
   (POP (GETR ORD)
        (T T)
        5))

(PP/
   (CAT PREP (T T)
        5
        (SETR PREP *)
        (TO PP/PREP)))

(PP/NP
   (POP (BUILDQ (PP (PREP +)
                    +)
                PREP NP)
        (T T)
        5))

(PP/PREP
   (PUSH NP/ ((NPSTART)
              T
              (NOT (ROLE SUBJ)))
        5
        (SETR NP *)
        (COND (FEATURES (ADDL LIFTLIST FEATURES)))
        (TO PP/NP)))

(QUANT/
   (PUSH NUMBER/ ((NUMBERSTART)
                  T T)
        5
        (SETR NUMB (NUMBOF *))
        (TO QUANT/QUANT)))

(QUANT/QUANT
   (TST UNIT-TST ((AND (NEQ (GETF NUMBER)
                            (QUOTE PL))
                       (MARKER UNIT *))
                  T)
        4
        (* "A SEVEN DAY TRIP")
        (SETR UNIT *)
        (TO QUANT/UNIT))
```

```
   (JUMP QUANT/UNIT (T T)
       3))

(QUANT/UNIT
   (POP (COND
         ((GETR ADV)
          (BUILDQ (COMP (ADV +)
                        (@ (NP (INTEGER +))
                           ))
                  ADV NUMB (COND
                    ((GETR UNIT)
                     (BUILDQ ((UNIT +))
                             UNIT))
                    (T NIL))))
         (T (BUILDQ (@ (INTEGER +)
                       )
                    NUMB
                    (COND ((GETR UNIT)
                           (BUILDQ ((UNIT +))
                                   UNIT)))))))
       (T T)
       5))

(R/
   (PUSH R/WHOSE ((RELSTART WHOSE)
                  T T)
       4
       (* "THE MAN WHOSE HOUSE BURNED")
       (SETRQ TYPE REL)
       (SETR WH *)
       (TO R/WH))
   (WRD (WHICH THAT WHO) (T T)
       4
       (SETRQ TYPE REL)
       (SETRQ WH **RELNP**)
       (TO R/WH))
   (WRD WHOM (T T)
       4
       (* **RELNP** IS NOT THE SUBJ - WILL BE PICKED
         UP LATER)
       (SETRQ TYPE REL)
       (SETRQ HOLDWH **RELNP**)
       (TO R/WH))
   (CAT PREP (T T)
       4
       (* "THE PERSON TO WHOM I WROTE")
       (SETR PREP *)
       (SETRQ TYPE REL)
       (SETRQ WH **RELNP**)
       (TO R/PREP)))
```

```
(R/NIL
   (PUSH NP/ ((NPSTART)
              (AND (NULLR VMODS)
                   (NULLR NEG))
              T)
         3
         (* "THE BOOK MANY PEOPLE READ")
         (SETRQ HOLDWH **RELNP**)
         (SETRQ TYPE REL)
         (SETR SUBJ *)
         (TO S/NP))
   (CAT V ((AND
            (NOT (AND (WRD BE (GETROOT * V))
                      (NOT (EQ * (QUOTE BEING)))))
            (OR (AND (GETF PASTPART)
                     (VPASSIVE *))
                (GETF PRESPART)))
           T)
         4
         (* "THE CHILD GOING TO BED" ,
           "A CLOTH SOAKED WITH WATER")
         (COND ((AND (GETF PASTPART)
                     (VPASSIVE *))
                (SETRQ SUBJ **RELNP**)
                (SETRQ TNS PAST)
                (SETRQ VOICE PASSIVE)
                (SETR AGFLAG T))
               ((GETF PRESPART)
                (SETRQ SUBJ **RELNP**)
                (SETRQ TNS PRESENT)
                (SETRQ VOICE ACTIVE)
                (SETRQ ASPECT PROGRESSIVE)))
         (SETR V *)
         (SETRQ WH **RELNP**)
         (SETRQ TYPE REL)
         (TO VP/V)))

(R/PREP
   (WRD (WHICH WHOM) (T T)
         3
         (ADDL VMODS (BUILDQ (PP (PREP +)
                                 **RELNP**)
                             PREP))
         (SETR WH NIL)
         (TO R/WH)))
```

```
(R/WH
   (PUSH NP/ ((NPSTART)
              T T)
         3
         (COND ((GETR WH)
                (SETR HOLDWH (GETR WH))))
         (SETR SUBJ *)
         (TO S/NP))
   (JUMP S/NP (T (AND (GETR WH)
                      (NULLR PREP)))
         4
         (SETR SUBJ (GETR WH))))

(R/WHOSE
   (WRD WHOSE (T T)
        5
        (SETR ART *)
        (ADDL LIFTLIST (QUOTE QDET))
        (SETR QDET T)
        (ADDL PREMODS (BUILDQ (POSS **RELNP**)))
        (TO NP/ART)))

(S/
   (PUSH PP/ ((PPSTART)
              T T)
         2
         (* "TO WHICH PLACE DID YOU GO" ,
           "TO A MAN THEY SAY 'CUTTY SARK'")
         (* IF THERE IS A QWORD OR QDET IN THE OBJECT OF
           THE PP IT BECOMES THE DS SUBJECT IN S/QP1)
         (SETR HOLDPP *)
         (COND ((GETF QDET)
                (SETRQ TYPE Q)))
         (TO S/))
   (JUMP S/Q ((QSTART)
              T)
         5
         (SETRQ TYPE Q))
   (WRD IF (T T)
        5
        (TO S/IF))
   (JUMP S/IMP ((CHECKF V UNTENSED)
                (NULLR TYPE))
         4)
   (JUMP S/DCL ((NOR (QSTART)
                     (CAT PREP))
                T)
         4
         (COND ((NULLR TYPE)
                (SETRQ TYPE DCL)))))
```

```
(S/AUX
   (CAT NEG (T (NULLR NEG))
        5
        (* "JOHN DIDN'T GO TO WASHINGTON")
        (SETRQ NEG NEG)
        (COND ((WRD DO MODAL)
               (SETR MODAL NIL)
               (* UNDOING DO-SUPPORT)))
        (TO S/AUX))
   (JUMP VP/V (T (OR (GETR THERE)
                     (AND (GETR SUBJ)
                          (PNCHECK (GETR SUBJ)
                                   (GETR PNCODE)))))
        4
        (* IF WE HAVE SUBJ, GO TO VP/V IF IT AGREES
          WITH VERB))
   (JUMP S/NO-SUBJ (T (NOR (GETR SUBJ)
                           (GETR THERE)))
        4
        (* "DID JOHN GO TO WASHINGTON")
        (* BECAUSE OF SUBJ-VERB INVERSION, WE HAVEN'T
          IDENTIFIED SUBJECT, SO GO TO S/NO-SUBJ TO
          FIND ONE)))

(S/DCL
   (PUSH NP/ ((NPSTART)
              T
              (NOT (ROLE OBJ)))
        3
        (* SUBJECT NP BEGINS DECL SENTENCE)
        (SETR SUBJ *)
        (TO S/NP))
   (WRD THERE (T T)
        4
        (* "THERE WAS A BOOK ON THE TABLE")
        (SETR THERE T)
        (* THERE-INSERTION HAS OCCURRED; WE MUST LATER
          FIND 'BE' OR 'EXIST')
        (TO S/NP)))

(S/HOW
   (CAT ADJ (T T)
        4
        (* "HOW BIG")
        (SETR HOLDADJ *)
        (TO S/NP)))
```

```
(S/IF
   (PUSH S/ ((SSTART)
             T
             (EQ (STYPE)
                 (QUOTE DCL)))
       5
       (SETR S1 (CONS (QUOTE S)
                      (CONS (QUOTE IF)
                            (CDDR (UNHASH *)))))
       (TO S/THEN)))

(S/IFTHEN
   (POP (GETR S2)
       (T T)
       5))

(S/IMP
   (CAT V ((GETF UNTENSED)
           T)
       5
       (* "GIVE ME ...")
       (* SET UP REGS FOR IMPERATIVE; GO TO VP/HEAD TO
         PICK UP POST-VERB CONSTITUENTS)
       (SETRQ TYPE IMP)
       (SETR SUBJ (BUILDQ (NP (DET)
                              (PRO YOU)
                              (FEATS (NU SG)))))
       (SETR V *)
       (SETR HEAD *)
       (SETRQ TNS PRESENT)
       (SETRQ VOICE ACTIVE)
       (TO VP/HEAD)))

(S/NO-SUBJ
   (PUSH NP/ ((NPSTART)
              (PNCHECK * (GETR PNCODE))
              (NOT (ROLE OBJ)))
       3
       (* IF SUCCESSFUL AND NP AGREES WITH VERB, IT IS
         SUBJ AND OUR INDECISION ABOUT WHQ IS
         RESOLVED
         (IT IS NOT SUBJ SO IT IS HELD TO BE PICKED
             UP AS OBJECT OR PREP OBJECT)
         ; UNDO DO-SUPPORT FOR S-V INVERSION)
       (SETR SUBJ *)
       (COND ((OR (GETR WH)
                  (GETR WHQ))
              (SETR HOLDWH (GETR WHQ))))
       (COND ((WRD DO MODAL)
              (SETR MODAL NIL)))
       (TO VP/V))
```

```
(WRD THERE (T (OR (NULLR WHQ)
                  (PNCHECK (GETR WHQ)
                           (GETR PNCODE))))
    4
    (* WE GET TO THIS STATE IF THERE WAS NO
      IDENTIFIABLE SUBJ BEFORE FIRST VERB. SUBJ
      MIGHT BE HERE IN STRING IF SUBJ-VERB
      INVERSION HAPPENED, OR IT MIGHT BE IN WHQ
      REG)
    (* ANYTHING IN WHQ MUST AGREE WITH VERB AND
      BECOME SUBJECT 'HOW MANY MEN WERE THERE...';
      IF WHQ IS EMPTY 'WERE THERE MANY MEN...'
      MOVE ON TO S/THERE)
    (COND ((GETR WHQ)
           (SETR SUBJ (GETR WHQ))
           (SETR WHQ NIL))
          ((GETR WH)
           (SETR SUBJ (GETR WH))
           (SETR WH NIL)))
    (SETR THERE T)
    (TO S/THERE))
(JUMP VP/V (T (AND (GETR WH)
                   (NOT (WRD HAVE V))
                   (PNCHECK (GETR WH)
                            (GETR PNCODE))))
    4
    (* 'THERE' AND NP DIDN'T WORK, BUT WH AGREES
      WITH VERB SO MAKE IT THE SUBJECT)
    (SETR SUBJ (GETR WH))
    (SETR WH NIL))
(JUMP VP/V (T (AND (GETR WHQ)
                   (PNCHECK (GETR WHQ)
                            (GETR PNCODE))))
    3
    (* DITTO FOR WHQ)
    (SETR SUBJ (GETR WHQ))
    (SETR WHQ NIL)))
```

```
(S/NP
   (POP (COND ((WRD Q TYPE)
               (BUILDQ (S NPQ +)
                       WHQ))
              (T (BUILDQ (S NPU +)
                         SUBJ)))
        (T (AND (NPUPOP (GETR SUBJ))
                (NOT (WRD REL TYPE))
                (NOT (GETR VMODS))))
        3
        (ADDL LIFTLIST (QUOTE NPU))
        (* "WHICH ONE")
        (* WE GET TO THIS STATE AFTER FIRST ATTEMPT AT
          FINDING AN NP, EITHER AS DCL SUBJECT OR AS A
          QUESTION WORD (WHQ)))
   (CAT MODAL (T T)
        4
        (* "JOHN MAY LEAVE" "WHICH CANDY CAN I EAT"
          "WILL JOHN LEAVE")
        (SETRQ VOICE ACTIVE)
        (SETR MODAL *)
        (SETR PNCODE (GETF PNCODE))
        (SETR TNS (GETF TNS))
        (TO S/AUX))
   (CAT V ((GETF TNS)
           T)
        5
        (* THIS IS THE ARC USUALLY TAKEN FROM THIS
          STATE. THE VERB IS EITHER THE FIRST WORD IN
          A QUESTION (IF WE JUMPED HERE FROM S/Q)
          OR IT FOLLOWS A SUBJ NP OR 'THERE'. IF WE
          ARE IN A WH QUESTION AND THE VERB IS NOT AN
          AUX ('WHO HIT JOHN')
          THEN THE WHQ IS THE SUBJ. IN ANY CASE, SAVE
          TENSE AND PERSON-NUMBER CODE)
        (SETRQ VOICE ACTIVE)
        (SETR V *)
        (COND ((AND (GETR WHQ)
                    (NOT (WRD (HAVE BE))))
               (SETR SUBJ (GETR WHQ))
               (SETR WHQ NIL)))
        (COND ((AND (GETR WH)
                    (NOR (MODAL)
                         (WRD (HAVE BE DO))))
               (SETR SUBJ (GETR WH))))
        (SETR PNCODE (GETF PNCODE))
        (SETR TNS (GETF TNS))
        (TO S/AUX)))
```

```
(S/Q
   (PUSH NP/ ((NPSTART)
              T
              (OR (EQ (PRONP)
                      (QUOTE ONE))
                  (NOT (PRONP))))
       4
       (* NORMALLY WILL BE REACHED ONLY WITH QWORD OR
         QDET FROM S/)
       (* ALLOWS "WHICH ONE(S)" BUT NOT "WHICH HE")
       (COND ((ROLE OBJ)
              (SETR HOLDNP (COPY *))
              (* HOLDWH?))
             (T (SETR WHQ (COPY *))))
       (TO S/NP))
   (WRD HOW (T T)
       5
       (SETR HOW *)
       (TO S/HOW))
   (CAT QWORD (T T)
       5
       (SETR TEMP (BUILDQ (NP (N   )
                              (FEATS (NU   )))
                          (COPY *)
                          (GETF NUMBER)))
       (COND ((ROLE SUBJ/OBJ)
              (SETR WHQ (GETR TEMP)))
             ((ROLE OBJ)
              (SETR OBJ (GETR TEMP)))
             (T (SETR HOLDWH (GETR TEMP))))
       (COND ((GETF ANAPHORIC)
              (SETRQ ANAPHORFLG T)))
       (TO S/NP))
   (CAT QADV (T (NULLR QADV))
       5
       (SETR QADV *)
       (TO S/Q))
   (JUMP S/NP ((OR (CAT MODAL)
                   (AND (CAT V)
                        (MEMB (GETROOT * V)
                              (QUOTE (BE HAVE DO WILL
                                      SHALL CAN
                                      MUST MAY)))))
               T)
       4))
```

```
(S/S
   (POP (GETR POPVAL)
        (T (AND (NULLR HOLDWH)
                (NULLR HOLDPP)
                (NULLR HOLDADJ)))
        5))

(S/THEN
   (PUSH S/ ((SSTART)
             T
             (FMEMB (STYPE)
                    (QUOTE (Q DCL))))
        5
        (* "IF WE DO X CAN WE DO Y"
          "IF WE DO X WE CAN DO Y")
        (* CHEK TYPE)
        (SETR S2 (BUILDQ (S + *)
                         S1))
        (TO S/IFTHEN))
   (WRD THEN (T (NULLR THEN))
        5
        (* "IF X THEN Y")
        (SETR THEN T)
        (TO S/THEN)))

(S/THERE
   (JUMP VP/V (T T)
        5))

(S/VP
   (JUMP S/S (T T)
        2
        (COND ((GETR HOLDPP)
               (* "BY TOMORROW JOHN WILL GO")
               (* PICKING UP VIRTUAL PP)
               (ADDL VMODS (GETR HOLDPP))
               (SETR HOLDPP NIL)))
        (COND ((AND (RFEAT COPULA V)
                    (GETR HOLDADJ))
               (* PICKING UP ADJ FOUND AT S/HOW)
               (SETR V (BUILDQ (ADJ +)
                               HOLDADJ))
               (SETR HOLDADJ NIL)))
        (SETR POPVAL (SBUILD))))
```

```
(TO/
   (WRD TO (T T)
        5
        (* "I WANT TO GO")
        (COND ((NULLR SUBJ)
               (SETRQ SUBJ **TOCOMP**)))
        (COND ((NULLR TYPE)
               (SETRQ TYPE TOCOMP)))
        (TO FOR/TO))

   (CAT NEG (T (NULLR NEG))
        2
        (* "I TOLD HIM NOT TO EAT THE CAKE")
        (SETR NEG *)
        (TO TO/)))

(VP/HEAD
   (PUSH NP/ ((NPSTART)
              (OR (VTRANS V)
                  (AND (NULLR THERE)
                       (RFEAT COPULA V)))
              (NOT (ROLE SUBJ)))
        4
        (* HERE WE PICK UP THE REGULAR OBJECT OF
          TRANSITIVE VERBS. NOTE THAT FOR A
          WHQ-QUESTION WITH 'BE' AS THE MAIN VERB
          (WHO IS THE LEADER)
          THE SUBJECT (THE LEADER)
          WAS PICKED AT STATE S/NO-SUBJ AT WHICH POINT
          THE WHQ WAS HELD. THUS WE DON'T LOOK FOR THE
          OBJECT ON THIS ARC, BUT RATHER ON THE
          SUBSEQUENT VIR ARC)
        (* WE ALSO GET PREDICATE NOMINATIVES OF
          COPULAS, AS IN 'THE TRIP COST $400', 'THE
          CONFERENCE LASTED 9 DAYS')
        (SETR OBJ *)
        (TO VP/NP))
   (PUSH COMPL/ ((COMPLSTART)
                 (RFEAT THATCOMP V)
                 (COMPTYPE (QUOTE THATCOMP)))
        2
        (* "I WISHED THAT IT WOULD HAPPEN")
        (SETR OBJ (BUILDQ (NP *)))
        (TO VP/VP))
```

```
(PUSH COMPL/ ((COMPLSTART)
              (RFEAT FORCOMP V)
              (COMPTYPE (QUOTE FORCOMP)))
     2
     (* "I WAITED FOR JOHN TO COME")
     (SETR OBJ (BUILDQ (NP *)))
     (TO VP/VP))

(PUSH TO/ ((TOCOMPSTART)
           (RFEAT TOCOMP V)
           T)
     3
     (* "I WANTED TO GO" "I WANTED JOHN TO GO"
       "IT IS NECESSARY TO GO")
     (* MUST DECIDE WHAT TO REPLACE DUMMY NODE WITH)
     (SETR
      OBJ
      (BUILDQ
        (NP  )
        (CHANGENODE
          *
          (QUOTE **TOCOMP**)
          (COND
            ((IT? (GETR SUBJ))
             (QUOTE (NP (DET)
                        (PRO SOMEONE)
                        (FEATS (NU SG)))))
            ((AND (RFEAT OBJLOW V)
                  (GETR OBJ))
             (GETR OBJ))
            (T (GETR SUBJ))))))
     (TO VP/VP))
(PUSH S/ ((SSTART)
          (SCOMP V)
          (NEQ (STYPE)
               (QUOTE NPU)))
     2
     (* LOOK FOR A COMPLEMENT WITH THAT DELETED -
       'HE KNEW I WOULD GO')
     (COND ((EQ (QUOTE THAT)
                (NP.DET (S.NP (UNHASH *))))
            (ABORT)))
     (SETR OBJ
           (BUILDQ
             (NP  )
             (CONS (QUOTE S)
                   (CONS (QUOTE THATCOMP)
                         (CDDR (UNHASH *))))))
     (TO VP/VP))
```

```
    (CAT PARTICLE (T (PARTICLEAGREE (GETR V)))
        3
        (* "I PICKED UP THE BABY")
        (SETR V (VERB-PARTICLE (GETR V)
                               *))
        (SETR HEAD (GETR V))
        (TO VP/HEAD))
    (JUMP VP/NP (T T)
        3
        (COND
         ((AND (OR (WRD BE V)
                   (WRD HAVE V)
                   (VTRANS V))
               (GETR HOLDWH))
          (* PICKING UP OBJECT WHICH WAS HELD)
          (SETR OBJ (GETR HOLDWH))
          (SETR HOLDWH NIL))
         ((OR (AND (RFEAT COPULA V)
                   (GETR HOW))
              (AND (RFEAT INTRANS V)
                   (OR (NOT (WRD BE V))
                       (GETR THERE)
                       (GETR QADV)))
              (AND (VTRANS V)
                   (OR (GETR OBJ)
                       (EQ (GETR VOICE)
                           (QUOTE PASSIVE)))))
          (* IF MAIN VERB IS INTRANSITIVE, THERE IS NO
             PREDICATE COMPLEMENT DIRECTLY TIED TO THE
             VERB, SO SKIP TO VP/NP))
         (T (ABORT)))))

(VP/NP
   (PUSH COMPL/ ((COMPLSTART)
                 (OR
                   (AND (OR (COMPTYPE (QUOTE FORCOMP))
                            (COMPTYPE (QUOTE THATCOMP))
                            )
                        (IT? (GETR SUBJ)))
                   (AND (COMPTYPE (QUOTE THATCOMP))
                        (GETR AGFLAG)))
                 T)
        2
        (COND ((GETR AGFLAG)
               (SETRQ AGFLAG NIL)))
        (SETR SUBJ (BUILDQ (NP *)))
        (* SEE COMMENT IN ANNGRAM)
        (TO VP/VP))
```

```
(PUSH TO/ ((TOCOMPSTART)
           (RFEAT TOCOMP V)
           T)
     3
     (* "I PERSUADED MAY TO GO" ,
       "I PROMISED MARY TO GO")
     (* SEE COMMENT IN ANNGRAM)
     (SETR CHNGE
          (COND
            ((IT? (GETR SUBJ))
             (QUOTE (NP (DET)
                        (PRO SOMEONE)
                        (FEATS (NU SG)))))
            ((AND (GETR OBJ)
                  (RFEAT OBJLOW V))
             (GETR OBJ))
            (T (GETR SUBJ))))
     (SETR TOCOMP (BUILDQ (NP  )
                          (CHANGENODE *
                                      (QUOTE
                                      **TOCOMP**)
                                      (GETR CHNGE))))
     (COND ((RFEAT COPULA V)
            (COND ((IT? (GETR SUBJ))
                   (* "IT COST $400 TO GO")
                   (SETR SUBJ (GETR TOCOMP)))
                  (T (* "THE TRIP COST $400 TO TAKE")
                     (ABORT))))
           (T (SETR OBJ (GETR TOCOMP))))
     (TO VP/VP))
(PUSH COMPL/ ((COMPLSTART)
              (AND (OR (COMPTYPE (QUOTE FORCOMP))
                       (COMPTYPE (QUOTE THATCOMP)))
                   (RFEAT (INDOBJ V)))
              T)
     2
     (* "I TOLD MARY THAT ...")
     (ADDL VMODS (BUILDQ (PP (PREP TO)
                             +)
                         OBJ))
     (SETR OBJ (BUILDQ (NP *)))
     (TO VP/VP))
```

```
   (PUSH NP/ ((NPSTART)
              (AND (RFEAT INDOBJ V)
                   (GETR OBJ)
                   (INDOBJ? (GETR OBJ)))
              (NOT (ROLE SUBJ)))
          3
          (* A NP CAN OCCUR IN THIS POSITION IF THE VERB
            CAN TAKE AN INDIRECT OBJECT. WHAT WE THOUGHT
            WAS THE OBJECT WAS REALLY THE INDIRECT
            OBJECT, AND THE NP HERE IS TO BE THE DIRECT
            OBJECT
            (I GAVE JOHN THE INFO --> I GAVE THE INFO TO
               JOHN))
          (* INDOBJ? TEST CHECKS THAT OBJECT IS HUMAN)
          (ADDL VMODS (BUILDQ (PP (PREP  )
                                  +)
                              (COND ((EQ (RFEAT INDOBJ V)
                                         (QUOTE FOR))
                                     (QUOTE FOR))
                                    (T (QUOTE TO)))
                              OBJ))
          (SETR OBJ *)
          (TO VP/VP))
   (JUMP VP/VP (T T)
          2)
   (CAT PARTICLE (T (PARTICLEAGREE (GETR V)))
          3
          (* A PARTICLE CAN OCCUR AFTER THE OBJECT:
            "LOOK THE INFORMATION UP")
          (SETR V (VERB-PARTICLE (GETR V)
                                 *))
          (SETR HEAD (GETR V))
          (TO VP/VP)))

(VP/V
   (PUSH NP/ ((NPSTART)
              (AND (GETR THERE)
                   (NULLR SUBJ)
                   (WRD (BE EXIST)
                        V))
              (NOT (PRONP)))
          4
          (* PUSHING FOR SUBJ OF THERE-INSERTED SENTENCE)
          (COND ((NOT (PNCHECK * (GETR PNCODE)))
                 (ABORT)))
          (SETR SUBJ *)
          (TO VP/V))
```

```
(CAT V (T T)
    5
    (* THE FIRST VERB CAN BE FOLLOWED BY OTHER
      VERBS TO FILL OUT THE
      PERFECT-PROGRESSIVE-PASSIVE AUXILIARY
      STRUCTURE. ADVERBS AND THE SUBJECT OF A
      THERE-INSERTED SENTENCE CAN BE INTERSPERSED
      BETWEEN THE VERBS--WE LOOP FOR THEM HERE)
    (* VERBS AFTER THE MAIN VERB MUST BE
      PARTICIPLES OR UNTENSED FORMS. IF A PAST
      PARTICIPLE, THE PREVIOUS VERB IN THE
      SEQUENCE MUST BE EITHER 'HAVE' (
        ASPECT=PERFECT)
      OR 'BE' (SENTENCE IS PASSIVE, IF POSSIBLE)
      A PRESENT PARTICIPLE MUST BE PRECEDED BY
      'BE' (ASPECT=PROGRESSIVE)
      OTHERWISE, THE CURRENT WORD MUST BE AN
      UNTENSED VERB AND THE REGISTER V MUST BE
      EMPTY (BECAUSE THE FIRST VERB WAS A MODAL)
      IF ANY OF THESE CONDITIONS IS SATISFIED, WE
      REPLACE THE VERB BY THE ROOT FORM OF THE
      CURRENT WORD.)
    (COND ((GETF PASTPART)
           (COND ((AND (WRD BE V)
                       (VPASSIVE *))
                  (SETRQ VOICE PASSIVE))
                 ((AND (NULLR ASPECT)
                       (WRD HAVE V))
                  (SETRQ ASPECT (PERFECT)))
                 (T (ABORT))))
          ((GETF PRESPART)
           (COND ((WRD BE V)
                  (ADDR ASPECT (QUOTE PROGRESSIVE)))
                 ((WRD POSS-ING TYPE))
                 (T (ABORT))))
          ((OR (NOT (GETF UNTENSED))
               (GETR V))
           (ABORT)))
    (SETR V *)
    (TO VP/V))
```

```
   (JUMP VP/HEAD ((NOT (CAT V))
                  (GETR SUBJ))
       4
       (* IF WE HAVE THE SUBJECT, WE CAN ASSUME THAT
         WE ALSO HAVE THE MAIN VERB
         (USUALLY, THIS WILL BE TRUE BECAUSE WE WOULD
                   HAVE LOOPED THROUGH THE FIRST ARC
                   AS LONG AS POSSIBLE)
         AND JUMP TO VP/HEAD TO LOOK FOR POST-VERB
         CONSTITUENTS. IF THE V REGISTER IS EMPTY
         (THE FIRST AND ONLY VERB WAS A MODAL)
         WE ABORT UNLESS THE MODAL WAS 'DO'--WE ALLOW
         'DO' TO BECOME THE MAIN VERB)
       (COND ((NULLR V)
              (COND ((WRD DO MODAL)
                     (SETRQ V DO)
                     (SETR MODAL NIL))
                    (T (ABORT)))))
       (SETR HEAD (GETR V)))
   (CAT ADV (T T)
       5
       (ADDL VMODS (BUILDQ (ADV *)))
       (TO VP/V)))

(VP/VP
   (PUSH PP/ ((PPSTART)
              T T)
       3
       (ADDL VMODS *)
       (TO VP/VP))
   (JUMP S/VP (T T)
       3)
   (CAT PREP (T (GETR WHQ))
       4
       (* "WHO DID YOU GIVE IT TO ON FRIDAY")
       (ADDL VMODS (BUILDQ (PP (PREP *)
                               +)
                           WHQ))
       (SETR WHQ NIL)
       (TO VP/VP)))
```

```
(LISPXPRINT (QUOTE (V: (DIGITS TEENS TENS CLASSES
/MONTH/ /WEEKDAY/))) T)

(DEFINEV

(DIGITS (1 2 3 4 5 6 7 8 9))

(TEENS (10 11 12 13 14 15 16 17 18 19))

(TENS (20 30 40 50 60 70 80 90))

(CLASSES (/MONTH/ /WEEKDAY/))

(/MONTH/ (JANUARY FEBRUARY MARCH APRIL MAY JUNE JULY
                AUGUST SEPTEMBER OCTOBER NOVEMBER
                DECEMBER))

(/WEEKDAY/ (MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY
                 SATURDAY SUNDAY))
)

(DECLARE: DONTCOPY
  (FILEMAP (NIL)))
STOP
```

APPENDIX III

Vocabulary and Syntax Classes

This appendix lists the 351 words which were in the dictionary of the BBN speech understanding system when the examples in Chapter Six were run. (A 569 word dictionary is now available, and a dictionary of more than 1000 words is being prepared.) After the listing of the words in the dictionary, they are broken into syntactic classes, with the number of words in each class indicated beside the class name. Finally, the syntactic features are given together with a list of the words which carry each feature. Features may be of the form FEATURE, (FEATURE), or (FEATURE VALUE).

This is not a listing of the dictionary as it appears to the system, but rather a derived cross reference which indicates the various parts of speech and syntactic features for each word.

APPENDIX III THE VOCABULARY

(A ABOUT ABOVE ACL ACOUSTICAL ACOUSTICS ADDITIONAL AFFORD AFTER AI AIR AIRPLANE ALL ALREADY ALSO AM AMHERST AMOUNT AN AND ANTICIPATE ANY ANYONE ANYWHERE APRIL ARE ARPA ARRANGE ASA ASK ASSOCIATION ASSUME ASSUMPTION AT ATTEND AUGUST AVAILABLE BATES BE BECAUSE BEEN BEFORE BEGINNING BEING BIG BILL BONNIE BOSTON BOTH BREAKDOWN BUDGET BUS BY CALIFORNIA CAN CANCEL CAR CARNEGIE CENT CHANGE CITY COLARUSSO COMPUTATIONAL CONFERENCE CONTINUE COSELL COST COSTING COSTS COUNTRY CRAIG CURRENT DATE DAVE DAY DECEMBER DENNIS DID DIVISION DO DOES DOLLAR DONE DUE@TO DURING EACH EIGHT EIGHTEEN EIGHTEENTH EIGHTH EIGHTY EITHER ELEVEN ELEVENTH END ENGLAND ENOUGH ESTIMATE-N ESTIMATE-V EVERY EVERYONE EXPECT EXPENSE EXPENSIVE FALL FARE FEBRUARY FEE FIFTEEN FIFTEENTH FIFTH FIFTY FIGURE FINAL FIRST FISCAL FIVE FOR FORTY FOUR FOURTEEN FOURTEENTH FOURTH GET GETS GETTING GIVE GIVEN GIVES GIVING GO GOES GOING GONE GOT GOTTEN GROUP HAD HALF HALVES HAS HAVE HAVING HE HER HIM HIS HOW HOWMANY HOWMUCH HUNDRED I IF IFIP IJCAI IN INTERNATIONAL IS IT JANUARY JERRY JOHN JULY JUNE K KNOW L.A. LAST LATE LEFT LINDA LINGUISTICS LIST LONDON LONG LOS@ANGELES LYN LYNN MADE MAKE MAKES MAKHOUL MAKING MANY MARCH MASSACHUSETTS MAY ME MEETING MEMBER MISCELLANEOUS MONEY MONTH MORE MOST MUCH MY NEED NEW@YORK NEXT NINE NINETEEN NINETEENTH NINETY NINTH NO NOT NOTE NOVEMBER NOW OCTOBER OF ON ONE ONLY OR OTHER OUT OVERHEAD PAJARRO@DUNES PARTICIPANT PAUL PAY PENNSYLVANIA PEOPLE PER PERSON PHONOLOGY PITTSBURGH PLACE PLEASE PLUS PRINT PROJECT-N PROJECT-V PURPOSE QUARTER REGISTRATION REMAIN REST REVISE RICH RICHARD ROUND@TRIP SANTA@BARBARA SCHEDULE SECOND SEND SENDING SENDS SENT SEPTEMBER SEVEN SEVENTEEN SEVENTEENTH SEVENTH SEVENTY SHE SINCE SITE SIX SIXTEEN SIXTEENTH SIXTH SIXTY SO SOCIETY SOME SOMEONE SPEECH SPEND SPENDING SPENDS SPENT SPRING ST.LOUIS START STATUS STOCKHOLM SUMMER SUPPOSE SUPPOSED SUPPOSITION SUR SWEDEN TAKE TAKEN TAKES TAKING TEN TENTH THAN THANK@YOU THAT THE THEIR THEM THERE THESE THEY THIRD THIRTEEN THIRTEENTH THIRTIETH THIRTY THIS THOSE THOUSAND THREE TIME TO TOO TOOK TOTAL TRAVEL TRIP TWELFTH TWELVE TWENTIETH TWENTY TWO UNANTICIPATED UNBUDGETED UNSPENT UNTAKEN US VARIOUS VISIT WANT WAS WASHINGTON WE WEEK WENT WERE WHAT WHEN WHERE WHICH WHO WHOM WHOSE WILL WINTER WISCONSIN WITH WITHIN WORKSHOP YEAR YES YOU)

APPENDIX III THE VOCABULARY

The syntactic categories:

(ADJ 23 (ACOUSTICAL ADDITIONAL AVAILABLE BIG COMPUTATIONAL CURRENT EACH ENOUGH EXPENSIVE FINAL FISCAL INTERNATIONAL LATE LEFT LONG MANY MISCELLANEOUS OTHER UNANTICIPATED UNBUDGETED UNSPENT UNTAKEN VARIOUS))

(ADV 18 (ALREADY ALSO ANYWHERE EITHER ENOUGH HOW LATE LONG MORE MOST MUCH NOW ONLY PLEASE SO THERE TOO YES))

(ART 8 (A AN NO THAT THE THESE THIS THOSE))

(CONJ 8 (AND BECAUSE BOTH IF OR PLUS SINCE SO))

(INTEGER 27 (EIGHT EIGHTEEN EIGHTY ELEVEN FIFTEEN FIFTY FIVE FORTY FOUR FOURTEEN NINE NINETEEN NINETY ONE SEVEN SEVENTEEN SEVENTY SIX SIXTEEN SIXTY TEN THIRTEEN THIRTY THREE TWELVE TWENTY TWO))

(MODAL 5 (CAN DID DO DOES WILL))

(N 70 (ACOUSTICS AIR AIRPLANE AMOUNT ASSOCIATION ASSUMPTION BEGINNING BREAKDOWN BUDGET BUS CAR CENT CHANGE CITY CONFERENCE COST COUNTRY DATE DAY DIVISION END ESTIMATE-N EXPENSE FALL FARE FEE FIGURE GROUP HALF HALVES LINGUISTICS LIST MEETING MEMBER MONEY MONTH MUCH NEED NOTE OVERHEAD PARTICIPANT PEOPLE PERSON PHONOLOGY PLACE PROJECT-N PURPOSE QUARTER REGISTRATION REST ROUND@TRIP SCHEDULE SITE SOCIETY SOME SPEECH SPRING STATUS SUMMER SUPPOSITION THANK@YOU TIME TOTAL TRAVEL TRIP VISIT WEEK WINTER WORKSHOP YEAR))

(NEG 1 (NOT))

(NPR 53 (ACL AI AMHERST APRIL ARPA ASA AUGUST BATES BILL BONNIE BOSTON CALIFORNIA CARNEGIE COLARUSSO COSELL CRAIG DECEMBER DENNIS ENGLAND FEBRUARY IFIP IJCAI JANUARY JERRY JOHN JULY JUNE L.A. LINDA LONDON LOS@ANGELES LYN LYNN MAKHOUL MARCH MASSACHUSETTS MAY NEW@YORK NOVEMBER OCTOBER PAJARRO@DUNES PENNSYLVANIA PITTSBURGH RICH RICHARD SANTA@BARBARA SEPTEMBER ST.LOUIS STOCKHOLM SUR SWEDEN WASHINGTON WISCONSIN))

(ORD 23 (EIGHTEENTH EIGHTH ELEVENTH FIFTEENTH FIFTH FIRST FOURTEENTH FOURTH LAST NEXT NINETEENTH NINTH SECOND SEVENTEENTH SEVENTH SIXTEENTH SIXTH TENTH THIRD THIRTEENTH THIRTIETH TWELFTH TWENTIETH))

APPENDIX III THE VOCABULARY

(PARTICLE 3 (IN ON OUT))

(POSS 5 (HER HIS MY THEIR WHOSE))

(PRECONJ 2 (BOTH EITHER))

(PREP 18 (ABOUT ABOVE AFTER AT BEFORE BY DUE@TO DURING FOR IN OF ON OUT PER SINCE TO WITH WITHIN))

(PRO 23 (ANYONE EVERYONE HE HER HIM I IT ME ONE SHE SOMEONE THAT THEM THESE THEY THIS THOSE US WE WHAT WHO WHOM YOU))

(QADV 2 (WHEN WHERE))

(QDET 5 (HOWMANY HOWMUCH WHAT WHICH WHOSE))

(QUANT 14 (ALL ANY BOTH EACH EITHER ENOUGH EVERY HOWMANY HOWMUCH MANY MORE MUCH OTHER SOME))

(QWORD 7 (HOW HOWMANY HOWMUCH WHAT WHICH WHO WHOM))

(SPECIAL 8 (DOLLAR HUNDRED K NO THAN THANK@YOU THOUSAND YES))

(V 85 (AFFORD AM ANTICIPATE ARE ARRANGE ASK ASSUME ATTEND BE BEEN BEGINNING BEING BUDGET CAN CANCEL CHANGE CONTINUE COST COSTING COSTS DID DO DOES DONE END ESTIMATE-V EXPECT FIGURE GET GETS GETTING GIVE GIVEN GIVES GIVING GO GOES GOING GONE GOT GOTTEN HAD HAS HAVE HAVING IS KNOW LAST LEFT LIST MADE MAKE MAKES MAKING NEED NOTE PAY PRINT PROJECT-V REMAIN REVISE SCHEDULE SEND SENDING SENDS SENT SPEND SPENDING SPENDS SPENT START SUPPOSE TAKE TAKEN TAKES TAKING TOOK TOTAL TRAVEL VISIT WANT WAS WENT WERE WILL))

APPENDIX III THE VOCABULARY

The syntactic features:

```
(INGCOMP (CANCEL CONTINUE GIVE START START))
(INTRANS (CONTINUE GO GO GO START))
(PASSIVE (CANCEL CONTINUE FIGURE GET GIVE MAKE  MAKE  SEND
       SEND START START TAKE) )
(QCOMP (CANCEL CONTINUE FIGURE GIVE SEND SEND TAKE))
(THATCOMP (END FIGURE))
(TRANS (CANCEL CONTINUE END FIGURE GET GIVE MAKE MAKE SEND
       SEND START START TAKE))
((ANAPHORIC) (WHICH))
((DETERMINED) (ANYONE I IT ME THAT THESE THIS THOSE US  WE
       WHO WHOM YOU))
((INDOBJ FOR) (MAKE MAKE))
((NUMBER PL) (HALVES PEOPLE THEM THESE THEY THOSE US WE))
((NUMBER SG) (ANYONE HE HER HIM I IT ME  ONE  SHE  SOMEONE
       THAT THIS WHO WHOM WHOM))
((NUMBER SG/PL) (WHAT WHAT WHICH WHO YOU))
((PARTICLEOF (LEAVE PUT ADD)) (IN))
((PARTICLEOF (ADD CONTINUE)) (ON))
((PARTICLEOF (LEAVE PRINT SEND  MAKE  CANCEL  FIGURE  FIND
       GO)) (OUT))
((PASTPART) (BEEN COST DONE GIVEN  GONE  GOTTEN  HAD  LEFT
       MADE SENT SPENT TAKEN))
((PNCODE 13SG) (WAS))
((PNCODE 1SG) (AM))
((PNCODE 3SG) (COST COST COST COST COSTS  DOES  DOES  GETS
       GIVES GOES HAS IS MAKES SENDS SPENDS TAKES))
((PNCODE ANY) (CAN CAN DID))
((PNCODE X13SG) (ARE WERE))
((PNCODE X3SG) (COST DO WILL))
((PRESPART) (BEGINNING BEING COSTING GETTING GIVING  GOING
       HAVING MAKING SENDING SPENDING TAKING))
((ROLE OBJ) (HER HIM ME THEM US WHOM WHOM))
((ROLE SUBJ) (HE I SHE WE))
((ROLE SUBJ/OBJ) (WHAT WHICH WHO WHO))
((TNS FUTURE) (WILL WILL))
((TNS PAST) (COST DID DID GOT HAD MADE SENT SPENT TOOK WAS
       WENT WERE))
((TNS PRESENT) (AM ARE CAN CAN COST COST  COST  COST  COST
       COSTS  DO  DOES  DOES  GETS GIVES GOES HAS IS MAKES
       SENDS SPENDS TAKES WILL))
((UNTENSED) (BE WILL)))) )
```

BIBLIOGRAPHY

[1] Aho, Alfred V. and Ullman, Jeffrey D. (1972)
THE THEORY OF PARSING, TRANSLATION, AND COMPILING (volume 1: Parsing)
Prentice-Hall Inc., Englewood Cliffs, N.J., 1972

[2] Allen, Jonathan and O'Shaughennsy, Douglas (1974)
Fundamental Frequency Contours of Auxiliary Phrases in English
Presented at the annual meeting of the Association for Computational Linguistics, July, 1974

[3] Anonymous (0000)
unfortunately some of the examples cited are part of the folklore and are not attributable to their originators (Answer to problem on page 12: "That which is, is. That which is not, is not. Is not that so?")

[4] Baker, James K. (1975a)
The DRAGON System - an Overview
IEEE Trans. on ASSP, 23:1 (Feb. 1975), p. 24-29

[5] Baker, James K. (1975b)
Stochastic Modeling as a Means of Automatic Speech Recognition
PhD thesis, Speech and Computer Science, Carnegie-Mellon Univ., April 1975

[6] Barnett, Jeffrey (1973)
A Vocal Data Management System
IEEE Trans. on Audio and Electroacoustics, AU-21:3 (June 1973) p. 185-188

[7] Bates, M. (1974)
The Use of Syntax in a Speech Understanding System
in Erman, IEEE Symposium on Speech Recognition, p. 226-233

[8] Bates, M. and Wolf, J. (1975)
Prosodics
technical report in BBN Speech Understanding System QPR no. 2, BBN Report no 3080 (AI Report no. 30), Bolt Beranek and Newman Inc, Cambridge, Mass. (May, 1975)

[9] Bobrow, D.G. (1963)
Syntactic Analysis of English by Computer -- A Survey
Proc. FJCC, 1963, p. 365-387

[10] Bobrow, D.G. and Fraser, J.B. (1969)
An Augmented State Transition Network Analysis Procedure
Proc. IJCAI, 1969, p. 557-567

[11] Bobrow, D.G., et.al. (1972)
TENEX, a Paged Time Sharing System for the PDP-10
CACM, 15:3 (March 1972) p. 135-143

[12] Bruce, Bertram (1976)
Pragmatics in Speech Understanding
to appear in Proc. IJCAI, 1976

[13] Carroll, Lewis (1960)
THE ANNOTATED ALICE
(Alice's Adventures in Wonderland)
with notes by Martin Gardner,
Bramhall House, N.Y. 1960

[14] Chomsky, Noam (1957)
SYNTACTIC STRUCTURES
Mouton, The Hague, 1957

[15] Chomsky, Noam (1964)
A Transformational Approach to Syntax
in Fodor and Katz, The Structure of Language, p. 211-245

[16] Chomsky, Noam (1965)
ASPECTS OF THE THEORY OF SYNTAX
MIT Press, Cambridge, Mass. 1965

[17] Coker, C.H. and Umeda, N. (1974)
Speech as an Error Correcting Process
Bell Laboratories, Murry Hill, N.J. (Aug. 1974)

[18] Cuadra, Carlos A. (ed.) (1968, 1974)
ANNUAL REVIEW OF INFORMATION SCIENCE AND TECHNOLOGY
Interscience, N.Y. 1968, 1974

[19] David E.E. and Denes P.B. (1972)
HUMAN COMMUNICATION
McGraw Hill, N.Y. (1972)

[20] Dixon, N.R. and Tappert C.C. (1972)
Derivation of Phonetic Representation by Combining Steady-State Transemic Classification in Automatic Recognition of Speech
in Conference Record, 1972 Conference on Speech Communication and Processing, IEEE catalogue no. 72 CH0596-7 AE

[21] Earley, J. (1970)
An Effecient Context-Free Parsing Algorithm
CACM, 13:2 (Feb 1970) p. 94-102

[22] Flanagan (1972)
SPEECH ANALYSIS, SYNTHESIS, AND PERCEPTION, 2nd ed.
Springer-Verlag, N.Y. 1972

[23] Fodor, Jerry A. and Katz, Jerrold J. (1964)
THE STRUCTURE OF LANGUAGE
Prentice-Hall Inc., Englewood Cliffs, N.J. (1974)

[24] Forgie, James W. (1973)
Speech Understanding Systems; Semiannual Technical Report
Lincoln Laboratory, Lexington, Mass. 30 Nov. 1973

[25] Forgie, James W. (1974a)
Overivew of the Lincoln System
in Erman,IEEE Symposium on Speech Recognition, p. 27

[26] Forgie, James W. (1974b)
Speech Understanding Systems; Semiannual Technical Summary
Lincoln Laboratory, Lexington, Mass., 31 May 1974

[27] Fromkin, V.A. (1971)
The Non-Anomalous Nature of Anomalous Utterances
Language, 47 (1971) p. 27-52

[28] Fu, K.C. (1975)
PATTERN RECOGNITION
Springer-Verlag, N.Y. (in press)

[29] Garrett, M., Bever, T.G., and Fodor, J.A. (1966)
The Active Uses of Grammar in Speech Perception
Percept. and Psychophysics, 1 (1966) p. 30-32

[30] Goldman-Eisler, F. (1972)
Pauses, Clauses, and Sentences
Language and Speech, 15 (1972) p. 103-113

[31] Grishman, R., et.al. (1973)
The Linguistic String Parser
AFIPS Conf. Proc. vol. 42, 1973, p. 427-434

[32] Harris, Zellig S. (1962)
STRING ANALYSIS OF SENTENCE STRUCTURE
Mouton and Co., The Hague, 1962

[33] Hopcroft, John E. and Ullman, Jeffrey D. (1969)
FORMAL LANGUAGES AND THEIR RELATION TO AUTOMATA
Addison-Wesley, Reading, Mass. 1969

[34] Hyde, S.R. (1968)
Automatic Speech Recognition: Literature Survey and Discussion
Report No. 45, Post Office Research Dept.
Dollis Hill, London, 1968
also in David and Denes, Human Communication

[35] Jakobson, R. (1972)
Verbal Communication
Sci. Am. 227 (1972) p. 72-80

[36] Joshi, A.K. et al (1972)
String Adjunct Grammars:
I. Local and Distributed Adjunction
II. Equational Representation, Null Symbols, and Linguistic Relevance
Info. and Control, 21 (1972), p. 93-116 and p. 235-260

[37] Kaplan, Ronald M. (1973)
A General Syntactic Processor
in Rustin, Natural Language Processing, p. 193-241

[38] Kay, Martin (1973)
The MIND System
in Rustin, Natural Language Processing, p. 155-188

[39] Kay, Martin and Sparck Jones, Karen (1971)
Automated Language Processing
in Cuadra, Ann. Review of Inf. Sci. and Tech., vol. 6, p. 141-166

[40] Klatt, D.H. and Stevens, K.N. (1973)
On the Automatic Recognition of Speech:
Implications from a Spectrogram-Reading
Experiment
IEEE Trans. on Audio and Electroacoustics, 21,
(June 1973) p. 210-217

[41] Klovstad, John W. (1975)
personal communication

[42] Klovstad, John W. and Mondshein, Lee F. (1974)
The CASPERS Linguistic Analysis System
in Erman, IEEE Symposium on Speech Recognition,
p. 234-240

[43] Kuno, S. and Oettinger, A. (1963)
Multiple Path Syntactic Analyzer
Information Processing 62, North-Holland, Amsterdam,
p. 306-312

[44] Lea, W., Medress M., and Skinner, T. (1972)
Prosodic Aids to Speech Recognition:
I. Basic Algorithms and Stress Studies
Univac Report no. PX7940, Oct. 1972

[45] Lea, W., Medress, M., and Skinner, T. (1973)
Prosodic Aids to Speech Recognition:
II. Syntactic Segmentation and Stressed
Syllable Location
Univac Report no. PX10232, April 15, 1973

[46] Lea, Wayne A. (1972)
Intonational Cues to the Constituent Structure and
Phonemics of Spoken English
PhD thesis, Purdue Univ., EE dept., June 1972

[47] Lehiste, Ilse (1967)
READINGS IN ACOUSTIC PHONETICS
MIT Press, 1967

[48] Lehiste, Ilse (1970)
SUPRASEGMENTALS
MIT Press, 1970

[49] Lehiste, Ilse (1975)
Role of Duration in Disambiguating Syntactically Ambiguous Sentences
Presented at the 89th Meeting of the Acoustical Society of America, University of Texas at Austin, 7-11 April, 1975
Abstract in JASA 57:supplement no. 1 (Spring 1975) S47

[50] Lipton, Richard J. and Snyder, Lawrence (1974)
On the Optimal Parsing of Speech
Research Report 37, (Oct. 1974) Dept. of Computer Science, Yale University, New Haven, Conn.

[51] McAllister, R. (1971)
Prediction Physical Aspects of English Stress
Speech Transmission Laboratory, QPR, April 15, 1971

[52] Meltzer, Bernard and Michie, Donald (eds) (1971)
MACHINE INTELLIGENCE 6
American Elsevier, New York, 1971

[53] Miller, Perry (1974)
A Locally Organized Parser for Spoken Input
CACM, 17:11 (Nov. 1974) 621-630
also PhD thesis, MIT EE Dept., 1973
also Lincoln Lab Tech Rept 503, 1973

[54] Mitre Corp. (1965)
English Preprocessor Manual
Mitre Corporation Report No. SR-132, May, 1965

[55] Nash-Webber, Bonnie (1974)
Semantic Support for a Speech Understanding System
IEEE Trans. on ASSP, 23:1 (Feb. 1975), p. 124-129

[56] Neely, R.B. (1973)
On the use of Syntax and Semantics in a Speech Understanding System
PhD. thesis, Stanford Univ, 1973
also Tech Rept. Comp. Sci. Dept. CMU, May, 1973

[57] Newell, A. et al (1973)
SPEECH UNDERSTANDING SYSTEMS:
Final Report of a Study Group
North-Holland, Amsterdam

[58] O'Malley, Michael H. and Herman, Steve J. (1972)
The Acoustic Features of Prosodies:
Survey and Bibliography
U. of Mich. Phonetics Lab, May 31, 1972

[59] Paxton, William H. (1974)
A Best-First Parser
in Erman, IEEE Symposium on Speech Recognition,
p. 218-225

[60] Paxton, William H. and Robinson, Ann E. (1973)
A Parser for a Speech Understanding System
Proc. 3rd IJCAI, Stanford, August, 1973, p. 216-222

[61] Petrick, S.R. (1965a)
A LISP Program for the Parsing of Sentences with
Respect to a Transformational Grammar
Proc. IFIP Congress, 1965, 528-529

[62] Petrick, S.R. (1965b)
A Recognition Procedure for Transformational
Grammars
PhD thesis, MIT Dept. of Modern Languages, May, 1965

[63] Petrick, S.R. (1966)
A Program for Transformational Syntactic Analysis
AFCRL-66-698 Air Force Cambridge Research Labs,
Bedford Mass, Oct 1966

[64] Petrick, S.R. (1973)
Transformational Analysis
in Rustin, Natural Language Processing, p. 27-41

[65] Pohl, Ira (1971)
Bi-Directional Search
in Meltzer and Michie, Machine Intelligence 6,
p. 127-140

[66] Reddy, D.R., Erman, L., and Neely, R.B. (1970)
The CMU Speech Recognition Project
IEEE System Sciences and Cybrenetics Conf.,
Pgh. Pa., 1970

[67] Reddy, D.R., Erman, L., and Neely, R.B. (1973a)
A Model and a System for Machine Recognition of
Speech
IEEE Trans. on Audio and Electro-acoustics, 21,
p. 229-238

[68] Reddy, D.R.,et.al. (1973b)
The HEARSAY Speech Understanding System:
An Example of the Recognition Process
Proc. 3rd IJCAI, Stanford, August, 1973, p. 185-193

[69] Ritea, H.B. (1974)
A Voice-Controlled Data Management System
in Erman, IEEE Symposium on Speech Recognition,
p. 28-31

[70] Rovner, P., Nash-Webber, B., and Woods, W. (1974)
Control Concepts in a Speech Understanding System
IEEE Trans. on ASSP, 32:1 (Feb. 1975), p. 136-140

[71] Rovner, P., et.al. (1974)
Where the Words are:
Lexical Retrieval in a Speech Understanding System
in Erman, IEEE Symposium on Speech Recognition,
p. 160-164

[72] Rustin, Randall (ed.) (1973)
NATURAL LANGUAGE PROCESSING
Algorithmics Press, N.Y. 1973

[73] Sager, Naomi (1973)
The String Parser for Scientific Literature
in Rustin, Natural Language Processing, p. 61-87

[74] Scholes, R.J. (1971)
ACOUSTIC CUES FOR CONSTITUENT STRUCTURE
Mouton and Co., The Hague, 1971

[75] Schwartz, R. and Makhoul, J. (1974)
Where the Phonemes are:
Dealing with Ambiguity in Acoustic-Phonetic
Recognition
IEEE Trans. on ASSP, 23:1 (Feb. 1975), p. 50-53

[76] Shirai, K. and Fujisawa, H. (1974)
An Algorithm for Spoken Sentence Recognition and its
Application to the Speech Input-Output System
IEEE Trans. on Systems, Man, and Cybrenetics,
Sept 1974, p. 475-479

[77] Simmons, Robert F. (1965)
Answering English Questions by Computer: a Survey
CACM, 8:1 (Jan 1965), p. 53-70

[78] Simmons, Robert F. (1970)
Natural Language Question-Answering Systems: 1969
CACM, 13:1 (Jan 1970), p. 15-30

[79] Teitelman, Warren (1974)
INTERLISP Reference Manual
Xerox Palo Alto Research Center, Palo Alto,
California, 1974

[80] Thorne, J.P., Bratley, P., and Dewar, H. (1968)
The Syntactic Analysis of English by Machine
in Michie, Machine Intelligence 3, p. 281-309

[81] Turn, Rein (1974)
The Use of Speech for Man-Computer Communication
The Rand Corp., Report R-1386-ARPA, Jan. 1974

[82] Vicens, P. (1969)
Aspects of Speech Recognition by Computer
Report CS-127, Comp. Sci. Dept., Stanford, 1969

[83] Yeager, M. and O'Malley, M.H. (1974)
Speech Styles and Pitch Contour on Negatives
J. Acoust. Soc. Am. 55, Supplement, p. 543

[84] Walker, Donald E. (1972)
Speech Understanding Research
SRI Annual Technical Report, Oct 1971 - Oct 1972

[85] Walker, Donald E. (1973a)
Speech Understanding Through Syntactic and Semantic
Analysis
Proc. 3rd IJCAI, Stanford, Aug. 1973, p. 208-215

[86] Walker, Donald E. (1973b)
Automated Language Processing
in Cuadra, Annual Review of Information Science and
Technology, vol 8
also SRI Technical Note 77 (June 1973)

[87] Walker, Donald E. (1974)
The SRI Speech Understanding System
in Erman, IEEE Symposium on Speech Recognition,
p. 32-37

[88] Warren, R. and Sherman, G. (1974)
Phonemic Restorations Based on Subsequent Context
Percept. and Psychophys., 16:1 (Aug 1974) p. 150-157

[89] Weinstein, C.J., et.al. (1974)
A System for Acoustic Phonetic Analysis of Continuous Speech
in Erman, IEEE Symposium on Speech Recognition, p. 89-100

[90] Wolf, Jared (1975)
Speech Recognition and Understanding
in Fu, Pattern Recognition (in press)

[91] Woods, W.A. (1969)
Augmented Transition Networks for Natural Language Analysis
Rep. CS-1, Comp. Lab. Harvard, 1969

[92] Woods, W.A. (1970)
Transition Network Grammars for Natural Language Analysis
CACM, 13:10 (Oct 1970) p. 591-606

[93] Woods, W.A. (1973)
An Experimental Parsing System for Transition Network Grammars
in Rustin, Natural Language Processing, p. 111-154

[94] Woods, W.A. (1974)
Motivation and Overview of BBN SPEECHLIS: An Experimental Prototype for Speech Understanding Research
IEEE Trans. on ASSP, 23:1 (Feb. 1975), p. 2-10

[95] Woods, W.A. (1975)
Syntax, Semantics, and Speech
in Reddy, Speech Recognition: Invited Papers Presented at the IEEE Symposium, Academic Press, (in press)
also BBN Report no. 3067 (AI Report no. 27), Bolt Beranek and Newman, Inc., Cambridge, Mass., April, 1975

[96] Woods, W.A. et. al. (1974)
Natural Language Communication with Computers, Final Report, Volume I, Speech Understanding Research at BBN
BBN Report 2976, Bolt Beranek and Newman Inc., Cambridge, Mass., Dec., 1974

[97] Woods, W.A., et.al. (1972)
The Lunar Sciences Natural Language Information System: Final Report
Report No. 2378, Bolt Beranek and Newman Inc., Cambridge, Mass.

[98] Yeager, M. and O'Malley, M.H. (1974)
Speech Styles and Pitch Contours on Negatives
J. Acoust. Soc. Am., 55, p. 543 (abstract)

[99] Zwicky, A.M., et.al. (1965)
The MITRE Syntactic Analysis Procedure for Transformational Grammars
FJCC, Nov-Dec 1965, p. 317-326